Real Estate Data Interchange Standard:

Real Estate Transaction Specification
Version 1.5

December 1, 2003

Second Edition

Table of Contents

List of Figures

List of Tables

SECTION

1

INTRODUCTION

1.1 Purpose

This specification is one of three documents defining the interchange of real estate information. The purpose of this document is to define a specification for the exchange of real estate property information, with the intent of eventually describing all interchangeable aspects of a real estate transaction. It defines a standard interface by which a client program may communicate with a property or other real estate data server. The specification defines a protocol for implementing transactions, and incorporates an Extensible Markup Language (XML) specification for general purpose interchange. The specification also provides for a compressed data interchange format and specification to allow interchange of machine-interpretable property information. This second document, the Real Estate Transaction DTD, defines the general structure of the XML DTD that can be used in transferring information from the server. Finally, this specification includes a DTD describing the metadata exchanged by endpoints using RETS.

1.2 Scope

This specification is intended to define only the minimum a product or service must do in order to be considered “compliant”. This specification is extensible and nothing in the specification precludes a vendor from adding data or functionality over and above that detailed here. However, when a function is provided or a data element is stored by a compliant system, it must offer access to the function or mechanism in a way that complies with the specification in order to be considered compliant.

1.3 Requirements

1.3.1 Required Features

This specification uses the same words as RFC 1123 [1] for defining the significance of each particular requirement. These words are:

MUST — This word or the adjective "required" means that the item is an absolute requirement of the specification. A feature that the specification states MUST be implemented is required in an implementation in order to be considered compliant.

SHOULD	This word or the adjective “recommended” means that there may exist valid reasons in particular circumstances to ignore this item, but the full implications should be understood and the case carefully weighed before choosing a different course. A feature that the specification states SHOULD be implemented is treated for compliance purposes as a feature that may be implemented.
MAY	This word or the adjective "optional" means that this item is truly optional. A feature that the specification states MAY be implemented need not be implemented in order to be considered compliant. However, if it is implemented, the feature MUST be implemented in accordance with the specification.

An implementation is not compliant if it fails to satisfy one or more of the MUST requirements for the protocols it implements. An implementation that satisfies all the MUST and all the SHOULD requirements for its protocols is said to be “unconditionally compliant”; one that satisfies all the MUST requirements but not all the SHOULD requirements for its protocols is said to be “conditionally compliant.”

Client and server implementations should generally follow the Internet protocol convention of being strict in what they generate, but tolerant in what they accept. However, in cases where tolerance of deviations from the specification could result in an incorrect interpretation of user data or intentions, implementers are urged to reject transactions rather than supplying possibly-incorrect defaults.

1.3.2 Compatibility with Prior Versions

The RETS 1.5 specification supercedes previous versions of the RETS specification. There is no *requirement* for a client or server that advertises itself as “compliant with RETS 1.5” to interoperate with earlier versions. However, client and server implementers are urged to support the prior version, RETS 1.0, in order to insure a smooth transition.

1.4 Terminology

Arguments	Tag/value pairs passed to a transaction as part of the Argument-List. The tag and the value are separated by an equal sign (“=”).
Argument-List	All the tag/value pairs for a request are Transfer-Encoded (see RFC 2616 [2]) and turned into a stream with each pair being separated by an ampersand (“&”) character. The tag/value stream is appended to the URI after a delimiting question mark (“?”) for the GET method. The tag/value stream is sent as the first entity body for the POST method. Transfer-Encoding MUST be used to indicate any transfer codings applied by an application to ensure safe and proper transfer of the Argument-List. The data and Arguments for a response are turned into a stream with each data record and Argument being separated by a carriage return/line feed.
Class	A subset of data elements within a Resource that share common metadata elements.

Client	The system requesting data. This may well be a server seeking to update itself from another server. The specification does not assume any particular kind of client.
Endpoint	Either a server or client.
Metadata	The set of data that describes data fields in detail.
Metadata Dictionary	The set of data that describes the available Metadata. This is the Meta-Metadata. It is used to determine the different classes of accessible data on the server and does not describe the fields within the those classes. It also defines what different types of searches are available (tax, open house, etc.).
Object	For purposes of RETS and its GetObject transaction, a collection of octets treated as a unit and associated with a unique resource element.
Optional	A compliant server or client is not required to include any field designated as optional. The specification states the action to be taken by a compliant system in the absence of an optional field. The fact that the specification designates a field as optional does not mean that the recipient of a transaction that is missing optional fields is required to provide all services that could be required if the field were present.
Required	A compliant server or client MUST include any field designated as required. A transaction that does not include every required field MUST be rejected by the recipient.
Resource	A collection of data having the external appearance of belonging to a single data base and being accessible for search or update via RETS transactions.
Resource Element	An individual record from a resource identified by a Resource Key.
Resource Key	The unique key that identifies a resource element.
Server	The system providing data (also referred to as the "host").
Session ID	A server-provided character string of up to 64 printable characters that uniquely identifies a session to a server. The contents are implementation-defined.
Request ID	A client-provided character string of up to 64 printable characters which uniquely identifies a request to a client. The contents are implementation-defined.
Standard-Name	The name of a data field as it is known in the Real Estate Transaction XML DTD.
System-Name	The name of a data field as it is known in the metadata.

SECTION

2

NOTATIONAL CONVENTIONS

2.1 Augmented BNF

This document expresses message layouts and character sequences in an augmented Backus-Naur Form (BNF) similar to that used by RFC 822 [4].

2.2 Typographic Conventions

Parsing constructs and examples are set in a `monospaced font`:

```
Server: Microsoft-IIS/4.0
```

In parsing constructs, textual elements that are required exactly as shown are indicated by **`boldface`** type., while textual elements that represent placeholders for actual data are indicated by a *`slanted font`*:

`Server:` *`server identifier`*

Entities designated by a textual definition contain that definition enclosed in angle brackets:

<any 8-bit sequence of data>

Atoms and primitive entities are indicated by *`ITALIC CAPS`*:

`1*64ALPHANUM`

Three nonprinting characters also have significance in some RETS constructs. These may be represented by special printing graphics:

→ Tab character, ASCII HT, an octet with a value of 9

↲ End of line

· Space character, where needed for clarity.

2.3 Rules

The following rules are used throughout this specification to describe basic parsing constructs. The US-ASCII coded character set is defined by ANSI X3.4-1986 [5].

Parsed entities are constructed combinations of atoms or other entities as defined below. Atoms may be combined and repeated to form longer constructs. When there are

constraints on the repetition of atoms, the constraints are expressed by a notation of the form:

m * *n*

where both *m* and *n* are integers. *m* represents the minimum allowed number of repetitions, and *n* represents the maximum. If *m* is omitted, it is presumed to be zero; if *n* is omitted, it is presumed to be infinite. For example, the syntactic construct

*1*64ALPHANUM*

means a string of *ALPHANUM*s containing at least 1 and at most 64.

When a parsing construct is represented by a string of entities, some of which are optional, the optional entities are enclosed in square brackets. For example, in the string

error-number [error-code]

the *error-number* entity is required, while the *error-code* entity is optional.

Alternation is indicated by the vertical bar. The entity description

ALPHA | DIGIT

means "either an *ALPHA* or a *DIGIT*".

2.4 Atoms and Primitive Entities

OCTET	::= <any 8-bit sequence of data>
CHAR	::= <any US-ASCII character (octets 0 - 127)>
UPALPHA	::= <any US-ASCII uppercase letter "A".."Z">
LOALPHA	::= <any US-ASCII lowercase letter "a".."z">
ALPHA	::= *UPALPHA* | *LOALPHA*
DIGIT	::= <any US-ASCII digit "0".."9">
ALPHANUM	::= *ALPHA* | *DIGIT*
SQLFIELDNAME	::= *ALPHA* *31*ALPHANUM* <except ANSI SQL 92 reserved words>
IDALPHA	::= *ALPHA* *NALPHANUM*
CTL	::= <any US-ASCII control character (octets 0 - 31) and DEL (127)>
CR	::= <US-ASCII CR, carriage return (13)>
LF	::= <US-ASCII LF, linefeed (10)>
SP	::= <US-ASCII SP, space (32)>
HT	::= <US-ASCII HT, horizontal-tab (9)>
<"> or **"**	::= <US-ASCII double-quote mark (34)>
NULL	::= <no character>
CRLF or ↵	::= *CR LF*
LWS	::= [*CRLF*] *1*(SP | HT)*
HEX	::= "A" | "B" | "C" | "D" | "E" | "F" | "a" | "b" | "c" | "d" | "e" | "f" | DIGIT

LHEX	::= "a" | "b" | "c" | "d" | "e" | "f" | *DIGIT*
TEXT	::= <any *OCTET* except *CTL*s, but including *LWS*>
PLAINTEXT	::= <any *OCTET* except *CTL*s>
TSPECIALS	::= "(" | ")" | "<" | ">" | "@" | "," | ";" | ":" | "\" | <"> | "/" | "[" | "]" | "?" | "=" | "{" | "}" | *SP* | *HT*
TOKEN	::= 1*<any *CHAR* except *CTL*s or *TSPECIALS*>
QUOTED-STRING	::= (<"> *(*QDTEXT*) <">)
QDTEXT	::= <any *TEXT* except <">>
DATE	::= Date using the format defined in RFC 1123.

SECTION

3

MESSAGE FORMAT

The Real Estate Transaction Specification is modeled on, and is compliant with HTTP/1.1. This specification does not require the implementation of persistent connections as defined in RFC 2616, however, it also does not preclude the use of them.

3.1 General Message Format

RETS messages consist of requests from a client to a server, answered by responses from the server and returned to the client. RETS requests use the generic message format of RFC 822, consisting of a start line, one or more header lines, an empty line, and zero or more body lines. The body may be null, depending on the message.

RETS responses use an RFC 822 header, but the body format is selected as appropriate for the content. For session control transactions such as Login (Section 4), the response body is a well-formed XML document containing the response arguments. For other transactions, the body may be MIME-encoded or may be a well-formed or valid XML document.

As in RFC 822, keywords in key-value pairs are not case-sensitive. The values, however, may be case-sensitive depending on context.

3.2 Request Format

Requests may take either of two forms. The form used is dependent on the Method. For the POST Method the post-request form is used. For the GET Method the get-request form is used. The major difference between the two forms is the location of the Argument-List. In the case of the get-request the Argument-List is appended to the Request-URI after a delimiting question mark ("?"). For the post-request the Argument-List is sent as the first entity body for the POST method.

```
post-request    ::=POST·Request-URI · HTTP-Version↲
                   *message-header
                   ↲
                   [Argument-List]

get-request     ::=GET·Request-URI [ ? Argument-List] ·HTTP-Version ↲
                   *message-header
                   CRLF
```

The *Request-URI*, *HTTP-Version* and *message-header* are defined in RFC 2616.

3.3 Header Field Format

A header field consists of three elements: a field name, a colon (":"), and a field value followed by a *CRLF*. The field value may be preceded or followed by any amount of *LWS*, though a single *SP* is preferred between the colon and the field value and no *LWS* is preferred after the field value; the *LWS* is not interpreted as part of the field value. The value itself may consist of any sequence of characters except CR or LF.

3.4 Required Client Request Header Fields

The header of any messages sent from the client MUST contain the following header fields:

`Accept` — This is a standard HTTP header field as defined in RFC 2616. Except for the GetObject Transaction, this value should be set to "*/*".

Example: `Accept: */*`

See Section 5.1 for more information on this field.

`User-Agent` — This header field contains information about the user agent originating the request. This is for statistical purposes, the tracing of protocol violations, and automated recognition of user agents for the sake of tailoring responses to avoid particular user agent limitations, as well as providing enhanced capabilities to some user-agents. All client requests MUST include this field. This is a standard HTTP header field as defined in RFC 2616.

User-Agent ::= **User-Agent:** · *product*

product ::= *token* [/ *product-version*]

product-version::= *token*

Example: `User-Agent: MLSWindows/4.00`

Product tokens should be short and to the point: use of them for advertising or other non-essential information is explicitly forbidden. Although any token character may appear in a product-version, this token SHOULD only be used for a version identifier (i.e., successive versions of the same product SHOULD only differ in the product-version portion of the product value). For more information about User-Agent see RFC 2616.

A server MAY advertise additional capabilities based on the client's User-Agent, and MAY refuse to proceed with the authorization if an acceptable User-Agent has not been supplied. A server MAY also choose to authenticate the client's identity cryptographically using RFC 2617; see Section 4.1, "Security" on page 1 for additional information.

`RETS-Version` — The client MUST send the RETS-Version. The convention used is a "<major>.<minor>" numbering scheme similar to the HTTP Version in Section 3.1 of RFC 2616. The version of a RETS message is indicated by a `RETS-Version` field in header of the message.

3.5 Optional Client Request Header Fields

`Authorization` — Authorization header field as defined in RFC 2617. See 4.1, "Security", as well as RFC 2617, for additional information.

`Cookie` — The implementation of this specification is intended to create a stateless system; however, because the user is required to log in there are at least two states. The Cookie mechanism MAY be used to provide a mechanism that can ensure that there are not multiple simultaneous sessions with a single username/password, if required by the server, and also to provide an added level of security. A new RETS-Session-ID cookie MAY be issued by the server at Login (see Section 4.5). This MUST be saved by the client application and sent in the client's HTTP request header as "`Cookie: RETS-Session-ID=`" to all subsequent requests. The `RETS-Session-ID` SHOULD be set to '0' for the initial Login.

cookie ::= **RETS-Session-ID=** *session-id*

session-id ::= *1*64ALPHANUM*

Example:

```
Cookie: RETS-Session-ID= AE872BC1DDFE7880DAD31233
```

`RETS-Request-ID` — A character string of printable characters which the client can use to identify this request. The contents are implementation-defined. If this field is included in a request from the client then the server MUST return it in the response.

RETS-Request-ID ::= *1*64ALPHNUM*

`Accept-Encoding` — A comma-separated list of MIME types indicating the content encoding schemes that the client is willing to accept. This is intended to support the use of compression in data returns; see section 3.9 for additional information.

*Accept-Encoding ::= 1*64ALPHNUM/1*64ALPHNUM[,1*64ALPHNUM/1*64ALPHNUM…]*

3.6 Response Format

The general server response to a request includes a *Status-Line*, one or more *header-line*s, a *CRLF* and a reply body. The *Status-Line* of a response consists of a status code and a (possibly empty) reason phrase.

server-reply ::= *Status-Line*
4**header-line*
CRLF
[*body-start-line*
response-body
rets-status
body-end-line]

Status-Line ::= *HTTP-Version* · *Status-Code* · *Reason-Phrase* *CRLF*

Status-Code ::= *1*4DIGIT*

The list of allowable *Status-Code*s can be found in RFC 2616. The more useful *Status-Code*s are provided in Section 3.10. Servers MUST use appropriate predefined status codes when communicating with the client. When an error is encountered a client MAY display both the status code and the associated Reason-Phrase in its communication with the user.

The *Status-Code* is intended to provide HTTP level errors to the client (Authorization, URI, etc.). Software level errors (search queries, invalid argument values, etc.) should be returned in the reply-code.

Reason-Phrase ::= *<TEXT, excluding CR/LF>

body-start-line ::= **<RETS** *1*SP* **ReplyCode=** *quoted-reply-code 1*SP* **ReplyText=** *quoted-string *SP*> *CRLF*

If a body is returned in the response then the body-start-line MUST be returned.

response-body ::= {*key-value-body* | *data*} *CRLF*

key-value-body ::= **<RETS-RESPONSE>***CRLF*(key = value CRLF)* **</RETS-RESPONSE>**

rets-status ::= **<RETS-STATUS** [*1*SP* **ReplyCode=***quoted-end-reply-code* *1*SP* **ReplyText=***quoted-string *SP*]**/>** *CRLF*

The *rets-status* MAY be included in the response if the ReplyCode or ReplyText given in the *body-start-line* becomes invalid during the creation of the response. If the server includes a *rets-status* in its reply, the client MUST use the ReplyCode and ReplyText from the rets-status rather than from the body-start-line.

body-end-line = **</RETS>** *CRLF*

If a *body-start-line* is returned in the response then the *body-end-line* MUST also be returned.

quoted-reply-code::= **"***1*5DIGITS***"**

The reply-code is included to provide a mechanism to pass additional information to the client in the event that the request is processed OK (Status-Code = 200) but some condition still exist that may require an action by the client. A value of '0' indicates success. Applicable reply-codes can be found under specific transactions.

quoted-end-reply-code= **"***1*5DIGITS***"**

The *end-reply-code* is included to provide a mechanism to pass additional information to the client in the event that the request being processed by the server errors before the request has been completed. This allows the server to start streaming out data before it has completed processing the request. A value of '0' indicates success, however the server SHOULD only send an *end-reply-code* if there is an error.

The valid <*key*>, <*value*> and <*data*> elements are defined in the Response Arguments section for each transaction.

An example server-reply where the reply body consists of key-value pairs:

```
HTTP/1.1 200 OK
Server: Microsoft-IIS/4.0
Date: Sat, 20 Mar 2002 12:03:38 GMT
Content-Type: text/xml
Cache-Control: private
RETS-Version: RETS/1.5
```

```
<RETS ReplyCode="0" ReplyText="SUCCESS">
<RETS-RESPONSE>
Key1=Value1
Key2=Value2
</RETS-RESPONSE>
</RETS>
```

3.7 Required Server Response Header Fields

The header of any messages sent from the server MUST contain the following header fields:

`Date` — The server MUST send the date using the format defined in RFC 1123. This is a standard HTTP header field as defined in RFC 2616.

Example: `Date: Sat, 20 Mar 1999 12:03:38 GMT`

The date/time stamp MUST be represented in Greenwich Mean Time (GMT), without exception.

`Cache-Control` — The RFC 2616 standard general-header field is used to specify directives that MUST be obeyed by all caching mechanisms along the request/response chain. The directives specify behavior intended to prevent caches from adversely interfering with the request or response. This field SHOULD be set to "private" for all transaction in this specification.

Example: `Cache-Control: private`

`Content-Type` — This is a standard HTTP header field as defined in RFC 2616. It specifies the media type of the underlying data. The server MUST return this field in all replies. For most replies this will be set to "**`text/xml`**". See Section 5.5 in the GetObject Transaction for exceptions and more information on this field.

Example: `Content-Type: text/plain`

`RETS-Version` — The server MUST send the RETS-Version. The convention used is a "<major>.<minor>" numbering scheme similar to the HTTP Version in Section 3.1 of RFC 2616. The version of a RETS message is indicated by a `RETS-Version` field in header of the message.

RETS-Version ::= **RETS-Version:** *version-info*

version-info ::= **RETS/** 1**DIGIT* **.** 1**DIGIT*

Example: `RETS-Version: RETS/1.5`

Applications sending request or response messages, as defined by this specification, MUST include a `RETS-Version` of "`RETS/1.5`". Use of this version number indicates that the sending application is compliant with this specification.

3.8 Optional Server Response Header Fields

`Content-Length` — The Content-Length entity-header field indicates the size of the message-body, in decimal number of octets. This is a standard

header field defined in RFC 2616 and is required for all requests containing a message-body not using Chunked transfer encoding.

`Transfer-Encoding` The Transfer-Encoding entity-header field when set to the Chunked value, indicates the size of the message-body is in the chunk stream. This is a standard header field defined in RFC 2616 and is required for all responses with a body not using `Content-Length` or a `Content-Type: Multipart` response.

`Content-Encoding` The Content Encoding entity-header field MAY be returned by the server if the client has included an AcceptEncoding header in its request () indicating that it can accept one or more compression types supported by the server. It is recommended that servers accept at least **`application/gzip`** (see 3.9, "Data Compression in RETS Transactions").

`Content-Encoding`::= *`1*64ALPHANUM / 1*64ALPHANUM`*

`RETS-Request-ID` The contents of the RETS-Request-ID field, if any, sent by the client in the request. If a RETS-Request-ID is included in a request from the client then the server MUST return it in the response.

`RETS-Request-ID` ::= *`1*64ALPHNUM`*

Server The server standard response-header field contains information about the software used to handle the request. The format of this field is the same as the User-Agent field in Section 3.4.

Example: `Server: Microsoft-IIS/4.0`

3.9 Data Compression in RETS Transactions

Clients and servers may choose to support data compression in data returned from the server. To indicate its willingness to accept compressed data, a client includes an `Accept-Encoding` header in its request. If the server supports one of the compression methods accepted by the client, it can include a `Content-Encoding` header in its response indicating the compression method it has chose.

Clients and servers choosing to implement compression SHOULD at least support GZip compression. This method is implemented by freely-available source code in a number of languages, as well as in several proprietary software development environments. A second freely-available alternative is BZIP. Clients and servers are free to choose other encoding methods as well.

3.10 General Status Codes

Any of the following status codes (in addition to the others provided in RFC 2616) may be returned by a server in response to any request:

Table 3-1 General Status Codes

Status	Meaning
200	Operation successful.
400	Bad Request The request could not be understood by the server due to malformed syntax.
401	Not Authorized Either the header did not contain an acceptable Authorization or the username/password was invalid. The server response MUST include a WWW-Authenticate header field.
402	Payment Required The requested transaction requires a payment which could not be authorized.
403	Forbidden The server understood the request, but is refusing to fulfill it.
404	Not Found The server has not found anything matching the Request-URI.
405	Method Not Allowed The method specified in the Request-Line is not allowed for the resource identified by the Request-URI.
406	Not Acceptable The resource identified by the request is only capable of generating response entities which have content characteristics not acceptable according to the accept headers sent in the request.
408	Request Timeout The client did not produce a request within the time that the server was prepared to wait.
411	Length Required The server refuses to accept the request without a defined Content-Length.
412	Precondition Failed Transaction not permitted at this point in the session
413	Request Entity Too Large The server is refusing to process a request because the request entity is larger than the server is willing or able to process.
414	Request-URI Too Long The server is refusing to service the request because the Request-URI is longer than the server is willing to interpret. This error usually only occurs for a GET method.
500	Internal server error. The server encountered an unexpected condition which prevented it from fulfilling the request.
501	Not Implemented The server does not support the functionality required to fulfill the request.
503	Service Unavailable The server is currently unable to handle the request due to a temporary overloading or maintenance of the server.
505	HTTP Version Not Supported The server does not support, or refuses to support, the HTTP protocol version that was used in the request message.

HTTP error status returns are only to be used for system level, transport syntax, and invalid transaction errors. RETS error status codes are used to indicate errors in the request arguments or the transaction processing.

SECTION

4

LOGIN TRANSACTION

A client MUST issue a login request prior to proceeding with any other request. The Login Transaction verifies all login information provided by the user and begins a RETS session. Subsequent session control may be mediated by HTTP cookies or any other method, though clients are required to support at least session control via HTTP cookies. Section 14 describes the session protocol in detail.

The server's response to the Login transaction contains the information necessary for a client to issue other requests. It includes URLs that may be used for other RETS requests, and may also contain identity and parameter information if required by the functions supported by the server.

4.1 Security

4.1.1 User Authentication

While this specification does not require the use of security — it permissible, for example, to operate a publicly-accessible RETS server — most operators of RETS servers will wish to authenticate users. A server that requires that users be authenticated MAY implement RFC 2617, HTTP Authentication. The use of at least digest authentication is strongly recommended.

4.1.2 Client Authentication

Some RETS servers may wish to authenticate the client application in addition to the user. This is provided for in RFC 2617 by the use of a `qop` value of `auth` or `auth-int` in the server's `WWW-Authenticate` header and the corresponding `cnonce` value in the client's `Authorization` header. A compliant RETS server MAY require that the client respond with a non-null `qop` value if the server has included a `qop` in its `WWW-Authenticate` header. However, this is a matter of server policy, not compliance: a client that merely complies with RFC 2617, which states that `qop` is optional in the Authorization header, is still considered compliant with this specification.

A RETS-compliant client can authenticate itself to a server though the use of a shared secret that becomes part of the computation of the RFC 2617 `cnonce`. The `cnonce` value is computed as

```
cnonce ::= H(User-Agent : Client-Password : RETS-Request-ID : nonce))
```

where *user-agent* is the value of the User Agent string (Section 3.4), *client-password* is the agreed shared secret, *RETS-request-ID* is the value of the RETS Request ID sent with this transaction (if one is used; see section 3.5), and *nonce* is the value of the nonce sent by the server in the last `WWW-Authenticate` header. If the client has not used a RETS Request ID, the value for computation of the `cnonce` should be the null string.

4.1.3 Data Security

Needs for secure HTTP transactions cannot be met by authentication schemes. For those needs, SSL or SHTTP are more appropriate protocols. A compliant server MAY support only HTTP-over-SSL. In this case, the server SHOULD listen on port 12109 rather than the standard RETS port, 6103.

4.2 Authorization Example

The following example assumes that a client application is trying to access the Login URI on the server using the POST method, and without using client authentication. The URI is "http://www.TheSite.com/login". Both client and server know that the username is "joesmith", and the password is "SuperAgent". The example also assumes the use of authentication using RFC 2617.

The first time the client requests the document, no Authorization header is sent, so the server responds with:

```
HTTP/1.1 401 Unauthorized
    WWW-Authenticate: Digest realm="Users@TheSite.com",
    nonce="dcd98b7102dd2f0e8b11d0f600bfb0c0"
    opaque="5ccdef346870ab04ddfe0412367fccba"
```

The client may prompt the user for the username and password, after which it will respond with a new request, including the following Authorization header:

```
Authorization: Digest username="joesmith",
    realm="Users@TheSite.com",
    nonce="dcd98b7102dd2f0e8b11d0f600bfb0c0",
    opaque="5ccdef346870ab04ddfe0412367fccba",
    uri="/login",
    response="13258d9b0bc217c9502b47e32dff8ee9"
```

4.3 Required Request Arguments

There are no required request arguments.

4.4 Optional Request Arguments

4.4.1 BrokerCode Argument

brokerCodeArgument ::= **BrokerCode** = *broker-code* , [*broker-branch*]

Some servers may support the scenario where a user belongs to multiple brokerages. If this is the case then the broker information (broker-code and broker-branch) must be input during login. If they are not included then the list of broker codes/branches is passed back to the client application through the response along with a "20012 Broker Code Required" reply-code.

broker-code ::= *1*24ALPHANUM*

broker-branch ::= *1*24ALPHANUM*

4.5 Optional Response Header Fields

In addition to the other Optional Server Response Header Fields specified in Section 3.5 the following response header field MAY be sent.

Note: Clients, but not servers, are required to implement cookie handling.

`Set-Cookie` The use of the Set-Cookie is required to provide a mechanism that, if required by the server, can guarantee that there are not multiple simultaneous sessions with a single username/password and also to provide an added level of security. A new RETS-Session-ID cookie is issued by the server at Login. This MUST be saved by the client application and sent in the HTTP header of any subsequent client requests during the session as "`Cookie: RETS-Session-ID=`".

Set-Cookie ::= **RETS-Session-ID=** *session-id SP* **path=/**

session-id ::= *1*64ALPHANUM*

Example:
```
Set-Cookie: RETS-Session-ID=AY872YOPOIPPOIP7880;
            path=/
```

Any server implementations that do not require the use of Session IDs should set the session-id in the response to '0'.

4.6 Login Response Body Format

The body of the login response has three basic formats when replying to a request. The simplest form is when there is an error:

<RETS *1*SP* **ReplyCode=** *quoted-reply-code 1*SP*
ReplyText= *quoted-string *SP* **/>** CRLF

The second case is where the user belongs to more than one broker and they have not provided broker information as part of the login. The reply contains a list of all brokerages the user belongs to.

<RETS ReplyCode = 20012 *1*SP* **ReplyText =** *quoted-string *SP* **>** *CRLF*
2(* **Broker =** *broker-code* [**,** *broker-branch*] *CRLF*)*
</RETS> *CRLF*

The third case is the normal "OK" response. In this case several arguments are passed back to the client in the response.

<RETS *1*SP* **ReplyCode=** *quoted-reply-code 1*SP*
ReplyText= *quoted-string *SP* **>** *CRLF*
<RETS-RESPONSE> ↵
member-name-key
user-info-key
broker-key
metadata-ver-key
min-metadata-ver-key
[*office-list-key*]
[*balance-key*]
[*timeout-key*]
[*pwd-expire-key*]
capability-url-list
</RETS-RESPONSE> ↵
[**<RETS-STATUS** [*1*SP* **ReplyCode=** *quoted-end-reply-code 1*SP* **ReplyText=**

quoted-string ***SP**]**/>**
</RETS> *CRLF*

4.7 Required Response Arguments

4.7.1 Broker

broker-key ::= **Broker =** *broker-code* [**,** *broker-branch*] *CRLF*

Broker information for the logged in user is returned to the client.

broker-code ::= *1*24ALPHANUM*

broker-branch ::= *1*24ALPHANUM*

These parameters are used in the validation routines of the Update Transaction (see Section 10 for more information).

4.7.2 Member Name

member-name-key ::= **MemberName =** *member-name CRLF*

The member's full name (display name) as it is to appear on any printed output.

member-name ::= *1*48TEXT*

4.7.3 Metadata Version Information

The metadata version keys indicate the current and minimum-acceptable versions of metadata.

metadata-ver-key::= **MetadataVersion =** *new-metadata-version CRLF*

This is the most current version of the metadata that is available on the server.

metadata-version ::=*1*2DIGITS* **.** *1*2DIGITS* [**.** *1*5DIGITS*]

It uses a "<major>.<minor>.<release>" numbering scheme.

min-metadata-ver-key::= **MinMetadataVersion =** *min-metadata-version CRLF*

This is the minimum version of the metadata that the host will support. If the version of the metadata being used by the client is less than this version the client MUST retrieve the newer metadata from the host.

min-metadata-version::=*1*2DIGITS* **.** *1*2DIGITS* [**.** *1*5DIGITS*]

It uses a "<major>.<minor>.<release>" numbering scheme.

The definition of the minumum version of the metadata is to permit clients to ignore non-essential changes to components such as help text and user-readable descriptions.

4.7.4 User information

user-info-key ::= **User =** *user-id* **,** *user-level* **,** *user-class* **,** *agent-code CRLF*

This key contains basic information about the user that is stored on the server. If a server does not support one of these fields then it MUST set the returned value to NULL.

user-id ::= *1*30ALPHANUM*

user-class ::= *1*30ALPHANUM*

user-level ::= *1*5DIGIT*

agent-code ::= *1*30ALPHANUM*

The agent-code is the code that is stored in the property records for the listing agent, selling agent, etc. In some implementations this may be the same as the user-id. The fields user-class and user-level are implementation dependent and may not exist on some systems, in which case, a value of NULL should be returned. These parameters are used in the validation routines of the Update Transaction (see Section 10 for more information).

4.7.5 Capability URL List

capability-url-list::= see Section 4.10 for format information

The server MUST return a capability list that includes at least a Search URL. The server MAY in addition return any of the other types in Section 4.10. If the server supports any of the additional functions (and the client is entitled to access the function by virtue of the supplied login information), it MUST provide URLs for those functions. The server MAY supply URLs in addition to those in Section 4.10 based on the user-agent. If it does, it MUST follow the format specified in Section 4.10.

4.8 Optional Response Arguments

4.8.1 Accounting Information

balance-key ::= **Balance** = *balance CRLF*

If the server supports an active billing account then this value SHOULD represent a user-readable indication of the money balance in the account.

balance ::= *1*32ALPHANUM*

4.8.2 Access Control Information

timeout-key ::= **TimeoutSeconds** = *1*5DIGIT*

The number of seconds after a transaction that a session will remain alive, after which the server will terminate the session automatically (e.g. invalidate the session-id). This is commonly referred to as the inactivity timeout. A server need not provide this capability; however, if it does use session timeouts in order to prevent monopolization of resources, it MUST inform the client of the timeout interval by returning this response field.

pwd-expire-key ::= **Expr** = *pwd-expr* , *expr-warn-per CRLF*

Indicates when a users password will expire. The parameter pwd-expr is a date in RFC 1123 format. And expr-warn-per is the number of days (*1*3DIGIT*) prior to expiration that the user should be warned of the upcoming password expiration. A expr-warn-per value of (-1) indicates that the password expiration is disabled.

4.8.3 Office List Information

office-list-key ::= **OfficeList** = *broker-code* [**;** *broker-branch*]
*(**,** *broker-code* [**;** *broker-branch*]) *CRLF*

If the logged in user is a company owner or manager they may have rights to login to multiple offices. The *office-list-key* is an enumeration of the offices to which the server will permit login.

broker-code ::= *1*24ALPHANUM*

broker-branch ::= *1*24ALPHANUM*

4.9 Well-Known Names

Some fields returned from the login are considered "Well-Known" and are used in the validation routines of the Update transaction. Those fields are as follows:

Table 4-1 Well-Known Names for Input Fields

Well-Known name	Input Return Field
.USERID.	user-id
.USERCLASS.	user-class
.USERLEVEL.	user-level
.AGENTCODE.	agent-code
.BROKERCODE.	broker-code
.BROKERBRANCH.	broker-branch

The client MUST assume a blank value for any well-known name for which the server does not supply an input field.

4.10 Capability URL List

The capability-url-list is the set of functions or URLs to which the login grants access. A capability consists of a key and a URL. The list returned from the server in the login reply has the following format:

```
[Action = action-URL CRLF]
[ChangePassword = change-password-URL CRLF]
[GetObject = get-object-URL]
Login = login-URL CRLF
[LoginComplete = login-complete-URL CRLF]
[Logout = logout-URL CRLF]
Search = search-URL CRLF
GetMetadata = get-metadata-URL CRLF
[Update = update-URL CRLF]
```

Table 4-2 Capability URL Descriptions

Parameter	Purpose
Action-URL	A URL on which the client MUST perform a GET immediately after login. This might include a bulletin or the notification of email. The client application SHOULD provide a means for the user to view the retrieved document.
Change-password-URL	A URL for the ChangePassword Transaction.
get-metadata-URL	A URL for the Get Metadata Transaction.
get-object-URL	A URL for the Get Object Transaction.

Table 4-2 Capability URL Descriptions

Parameter	Purpose
Login-URL	A URL for the Login Transaction. The client software should use this URL the next time it performs a Login. If this URL is different that the one currently stored by the client the client, MUST update the stored one to the new one. This provides a mechanism to move the Login server.
Login-complete-URL	RESERVED
Logout-URL	A URL for the Logout Transaction.
Search-URL	A URL for the Search Transaction.
Update-URL	A URL for the Update Transaction.

The URLs in the capability-url-list may be specified in any order. In addition, the table is extensible; servers may define additional transactions for clients to access. If a transaction is offered only to particular user agents, the keys for those additional transactions MUST begin with the user-agent token, followed by a dash "-", followed by an implementation-defined function name.

additional-transaction::= *user-agent-token* - *function-name*

user-agent-token ::= token portion of the user-agent (Section 3.4)

function-name ::= *1*ALPHA*

Example: `MLSWindows-special = /special_function`

A compliant client need not recognize any transaction that is not included in this specification. If some extended transactions are offered to any user-agent, the keys for those transactions MUST begin with an "X" followed by a dash, followed by an implementation-defined function name. Server implementers who implement potentially-unrestricted extension transactions are urged to register their keys and service descriptions on the RETS web site to encourage wider adoption.

URLs may either be absolute or relative. Any URL beginning with a " / " is considered to be relative and is relative to the Login-URL.

URLs MUST be URL-encoded per RFC 2396.

4.11 Reply Codes

Table 4-3 Valid Reply Codes for Login Transaction

Reply Code	Meaning
0	Operation successful.
20003	Zero Balance The user has zero balance left in their account.
20004 thru 20011	RESERVED
20012	Broker Code Required The user belongs to multiple broker codes and one must be supplied as part of the login. The broker list is sent back to the client as part of the login response (see section 4.6).
20013	Broker Code Invalid The Broker Code sent by the client is not valid or not valid for the user
20014 thru 20019	RESERVED

Note: RETS does not require that a server maintain user accounts.

Table 4-3 Valid Reply Codes for Login Transaction (continued)

Reply Code	Meaning
20022	Additional login not permitted There is already a user logged in with this user name, and this server does not permit multiple logins.
20036	Miscellaneous server login error The quoted-string of the body-start-line contains text that SHOULD be displayed to the user
20037	Client password invalid. The server requires the use of a client password (section 4.1.2), and the client either did not supply the correct client password or did not properly compute its challenge response value.
20050	Server Temporarily Disabled The server is temporarily offline. The user should try again later

SECTION

5

GETOBJECT TRANSACTION

The GetObject transaction is used to retrieve structured information related to known system entities. It can be used to retrieve multimedia files and other key-related information. Objects requested and returned from this transaction are requested and returned as MIME media types. The message body for successful retrievals contains only the objects in the specified MIME media type. Error responses follow the normal response format (section 3.10).

5.1 Required Client Request Header Fields

In addition to the Required Client Request Header Fields specified in Section 3.4, the header of any messages sent from the client MUST contain the following header fields:

`Accept` The client MUST request a media <type> using the standard HTTP Accept header field. Media-type formats (subtypes) are registered with the Internet Assigned Number Authority (IANA) and use a format outlined in RFC 2045 [8]. When submitting a request the client MUST specify the desired type and format. If the server is unable to provide the desired format it SHOULD return a "406 Not Acceptable" status. However, if there are no objects of any <subtype> available for the requested object the server SHOULD return "404 Not Found." The format of the Accept field is as follows:

Accept ::= **Accept:** *type* **/** *subtype* [**;** *parameter*]
*(** **,** *SP type* **/** *subtype* [**;** *parameter*])

type ::= ***** | **text** | **image** | **audio** | **video**

subtype ::= ***** | <A publicly-defined extension token that has been registered with IANA>

parameter ::= **q =** < qvalue scale from 0 to 1 >

A compliant server MUST support at least text/plain, text/xml and, if images are supported, image/jpeg. The more common <types> and <subtypes> are as follows:

```
text/plain          image/gif
text/xml            image/jpeg
```

```
text/html           image/tiff
video/mpeg          audio/basic
video/quicktime
```

A more complete list is available at:

ftp.isi.edu/in-notes/iana/assigments/media-types

The qvalue is used to specify the desirability of a given media type/format, with "1" being the most desirable, "0" being the least desirable, and a range in between. The default qvalue is "1".

Example:
```
Accept: image/jpeg, image/tiff;q=0.5,
  image/gif;q=0.1
```

Verbally, this would be interpreted as "image/jpeg is the preferred media type, but if that does not exist, then send the image/tiff entity, and if that does not exist, send the image/gif entity."

The types supported by the server are defined in the Metadata Dictionary as defined in section 11.4.1.

5.2 Optional Client Request Header Fields

The GetObject transaction has no optional request header fields.

5.3 Required Request Arguments

`Resource` <A Resource defined in the metadata Dictionary (see Section 11.2.2)>

The resource from which the object should be retrieved is specified by this entry. For more information see 5.9 . The Resource MUST be a resource defined in the metadata (section 11.4.1).

Type ::= <A type defined in the metadata (see Section 11.4.1)>

The grouping category to which the object belongs. The Type MUST be an ObjectType defined in the Object metadata for this Resource. For more information see section 11.4.1.

ID ::= *resource-set**(, *resource-set*)

resource-set ::= *resource-entity*[: *object-id-list*]

resource-entity ::= *1*ALPHANUM*

object-id-list ::= *object-id* *(: *object-id*)

object-id ::= *1*ALPHANUM*

The identity of the object. For objects, the resource-entity is a value (e.g., MLS number, AgentID) from the KeyField of the Resource for which the object is to be retrieved.

The object-id is the particular object to be retrieved. This parameter MUST be numeric; objects are assumed to be stored sequentially on the host beginning with an object-id of "1". If the object-id is 0 (zero or not provided), the designated preferred object of the given type is returned. If the object-id is set to "*" then all objects corresponding to the resource-

entity are returned. This parameter can be used to specify the photo number e.g. a value of "3" would indicate photo number 3.

Note: If multiple resource-entities or object-ids are sent then the host MUST respond with a multipart MIME response.

5.4 Optional Request Arguments

5.4.1 Location

Location **0 | 1**

This parameter indicates whether the object or a URL to the object should be returned. This is used to provide access to the semi-permanent storage location of information for access outside of the transaction (e.g. for use in email to a customer). If this parameter is set to "1" the server MAY return a URL to the given object. The default is "0". The server MAY support this functionality (Location="1") but MUST support Location="0". In other words, some servers may store the objects in a database or generate them dynamically. Therefore, it may not be possible for those servers to return a URL to the requested object. In these cases the server MAY choose not to support Location="1". However, all servers MUST support a method to get the object and therefore, MUST support the case where Location="0".

5.5 Required Server Response Header Fields

In addition to the other Required Server Header Fields specified in Section 3.7 the following response header fields are required.

`Content-Type` The media type of the underlying data. The server MUST return this field in all replies. Additionally, this field MUST be returned as part of the header for each body part. This field MUST be set to the type of media returned. See Section 5.1 for more information on <*type*> and <*subtype*>.

Content-Type ::= **Content-Type:** *type/subtype*

Example: `Content-Type: image/jpeg`

If the client has requested multiple IDs, the server MAY return a multipart message. If it does, it MUST return a Content-Type of "multipart/parallel" along with a boundary delimiter in the response header. See Section 5.11 for more information on multipart responses.

Example: `Content-Type: multipart/parallel; boundary=AAABBBCCC`

headerContent ID An ID for the object. This field MUST be returned as part of the header for each body part in a multipart response.

Content-ID ::= **Content-ID:** **64<TEXT, EXCLUDING CR/LF>*

Example: `Content-ID: 123456`

headerObject-ID The object number being returned. This field MUST be returned as part of the header for each body part in a multipart response.

Object-ID ::= **Object-ID:** *1*5DIGIT*

Example: `Object-ID: 2`

headerMIME-Version — All responses MUST include a MIME-Version of "1.0" in the response header.

Example: `MIME-Version: 1.0`

5.6 Optional Server Response Header Fields

In addition to the other Optional Server Header Fields specified in Section 3.8 the following response header fields are also optional.

5.6.1 Location

Location — If the client has submitted a request with "Location=1" the header of the response MUST contain the Location header field. If the server does not support this functionality then "Location:" without a URI should be returned.

Location ::= **Location:** *URL*

Example: `Location: http://www.TheSite.com/pic/123456.jpg`

5.6.2 Description

headerDescription — A text description of the object.

Description ::= **Content-Description:** **64<TEXT, EXCLUDING CR/LF>*

Example: `Content-Description: Front View`

5.7 Required Response Arguments

There are no required response arguments.

5.8 Optional Response Arguments

There are no optional response arguments.

5.9 Metadata

To retrieve objects the client MAY first retrieve the metadata that describes the Resources and Objects that are available with the GetMetadata transaction described in section 12. A full description of the Metadata Dictionary is provided in Section 11.

5.10 Resources

RETS does not require that any particular type of data be made available by a server. However, a server MUST use a standard well-known name under which to make its data available if a suitable well-known name is defined in the standard.

5.11 Multipart Responses

In the case where the client has requested multiple IDs, the server MUST return a multipart response. In the case of multipart responses, in which one or more different sets of data are combined in a single body, a “multipart” media type field must appear in the entity's header. The body must then contain one or more body parts, each preceded by a boundary delimiter line, and the last one followed by a closing boundary delimiter line. After its boundary delimiter line, each body part then consists of a header area, a blank line, and a body area.

The Content-Type field for multipart entities requires one parameter, “boundary”. The boundary delimiter line is then defined as a line consisting entirely of two hyphen characters (“-”, decimal value 45) followed by the boundary parameter value from the Content-Type header field, optional linear whitespace, and a terminating CRLF.

The CRLF preceding the boundary delimiter line is conceptually attached to the boundary so that it is possible to have a part that does not end with a CRLF (line break). Body parts that must be considered to end with line breaks, therefore, must have two CRLFs preceding the boundary delimiter line, the first of which is part of the preceding body part, and the second of which is part of the encapsulation boundary.

The boundary delimiter MUST NOT appear inside any of the encapsulated parts, on a line by itself or as the prefix of any line. It must be no longer than 70 characters, not counting the two leading hyphens. Because boundary delimiters must not appear in the body parts being encapsulated, a user agent must exercise care to choose a unique boundary parameter value. The boundary parameter value in the example above could have been the result of an algorithm designed to produce boundary delimiters with a very low probability of already existing in the data to be encapsulated without having to prescan the data.

The boundary delimiter line following the last body part is a distinguished delimiter that indicates that no further body parts will follow. Such a delimiter line is identical to the previous delimiter lines, with the addition of two more hyphens after the boundary parameter value.

Example:

```
HTTP/1.1 200 OK
Server: Microsoft-IIS/4.0
Date: Sat, 22 OCT 2000 12:03:38 GMT
Cache-Control: private
RETS-Version: RETS/1.0
MIME-Version: 1.0
Content-type: multipart/parallel; boundary="simple boundary"

--simple boundary
Content-Type: image/jpeg
Content-ID: 123456
<binary data>
--simple boundary
Content-Type: image/jpeg
Content-ID: 123457
<binary data>

--simple boundary--
```

5.12 Reply Codes

Table 5-1 GetObject Reply Codes

Reply Code	Meaning
20400	Invalid Resource The request could not be understood due to an unknown resource.
20401	Invalid Type The request could not be understood due to an unknown object type for the resource.
20402	Invalid Identifier The identifier does not match the KeyField of any data in the resource.
20403	No Object Found No matching object was found to satisfy the request.
20406	Unsupported MIME type The server cannot return the object in any of the requested MIME types.
20407	Unauthorized Retrieval The object could not be retrieved because it requests an object to which the supplied login does not grant access.
20408	Resource Unavailable The requested resource is currently unavailable.
20409	Object Unavailable The requested object is currently unavailable.
20410	Request Too Large No further objects will be retrieved because a system limit was exceeded.
20411	Timeout The request timed out while executing
20412	Too many outstanding requests The user has too many outstanding requests and new requests will not be accepted at this time.
20413	Miscellaneous error The server encountered an internal error.

SECTION

6

LOGOUT TRANSACTION

The Logout transaction terminates a session. Except in cases where connection failure prevents it or the user has requested an immediate shutdown of the client, the client SHOULD send the Logout transaction. If the client sends a Logout transaction, the server MUST attempt to send a response before terminating the session.

The server MAY send accounting information back to the client in the response to this transaction. The client is not required to display or otherwise process the accounting information.

6.1 Required Request Arguments

There are no required request arguments.

6.2 Optional Request Arguments

There are no optional request arguments.

6.3 Required Response Arguments

There are no required response arguments.

6.4 Optional Response Arguments

ConnectTime — The amount of time that the client spent connected to the server, specified in seconds.

```
ConnectTime    ::=   1*9DIGITS
```

Billing — If the server supports an active billing account, this is total amount billed for this session, specified as TEXT which is implementation-defined

```
Billing        ::=   *<TEXT, EXCLUDING CR/LF>
```

SignOffMessage — Any text. The client MAY display this message, if the server includes it in the response. Servers should not expect, however, that users would read or see the message, since communication

failure may make it impossible for the client to receive the Logoff response.

SignOffMessage ::= *<*TEXT, EXCLUDING CR/LF*>*

SECTION

7

SEARCH TRANSACTION

The Search transaction requests that the server search one or more searchable databases and return the list of qualifying records. The body of the response contains the records matching the query, presented in the requested format. The data can be returned in one of three formats: COMPACT, COMPACT-DECODED or STANDARD-XML.

7.1 Search Types

The server MUST support at least one type of search. The types of searches supported by the server are specified in the metadata. Each of these searches may by conducted against different databases or tables depending on the server implementation.

Some searches are specified by well-known names. If a server implementation supports one of these searches, it SHOULD use the well-known name to describe that search.

ActiveAgent	An ActiveAgent Search is a search against the Agent/Member database/table. This search only returns active agents. These are agents that are currently authorized to access the server (paid-up, not retired, etc.)
Agent	An Agent Search is a search against the Agent/Member database/table. It is used for retrieving information about the agents.
History	A History Search is a search against the history database/table.
Office	An Office Search is a search against the office database/table. It is used for retrieving information about the offices.
OpenHouse	An OpenHouse Search is a search against the Open House database/table.
Property	A Property Search is a search against the property database/tables. If the server supports a cross-property search then the metadata will define a class to use to perform the cross-property search.
Prospect	A Prospect Search is a search against the Prospect database/tables. A Prospect Search is used to retrieve prospect information from the server database.

Tax	A Tax Search is a search against the public records database/tables. Many systems have multiple public record databases. Each public record database is assigned a class number. This class number is used by the client application when submitting a search to distinguish which database the search will be conducted against.
Tour	A Tour Search is a search against the Tour database/table.

Note RETS does not require that a server support any particular search, but it does require that a server support at least one search. The user or maintainer of a server is responsible for deciding which resources should be made searchable.

7.2 Search Terminology

7.2.1 Field Delimiter

A server may designate a particular OCTET to be used as a delimiter for separating entries in both the COLUMNS list and the DATA returned using the COMPACT and COMPACT-DECODED formats. The octet should be chosen to avoid the need to escape data within a record

field-delimiter ::= *HEX HEX*

7.2.2 Field Name

A field is the keyword or code that the server uses to identify a particular column in the database table. Each field may be either a System-Name, as defined in the metadata, or a Standard-Name, as defined in the Real Estate Transaction XML DTD. The server MUST accept either set of names interchangeably.

field ::= *SQLFIELDNAME*

Any valid data for a field.

field-data ::= **TEXT*

7.2.3 Record Count

This value indicates the number of records on the server matching the search criteria sent in the search query.

record-count ::= *1*9DIGITS*

Note that this value may be greater than the number of records returned, if the server has limited the size of the return for any reason.

7.2.4 Other terms

XML-data-record ::= <A data record as defined by the RETS Data XML DTD>.

7.3 Required Request Arguments

7.3.1 Search Type and Class

The SearchType and Class arguments specify the data that the server is to search.

SearchType ::= *ResourceID*

The type of search to perform as discussed in Section 7.1 and defined in the Metadata (see section 11.2.2).

Class :: = *1*32ALPHANUM*

This parameter is set to a value that represents the class of data within the SearchType, taken from the Class metadata (section 11.3.1). If the resource represented by the SearchType has no classes, the Class parameter will be ignored by the server and MAY be omitted by the client. If the client does include the Class parameter for a classless search, the value SHOULD be the same as the *ResourceID* in order to insure forward compatibility.

7.3.2 Query Specification

The specification consists of the query itself together with a designation of the query language.

Query ::= <The query to be executed by the server>

The query is specified in the language described in Section 7.7.

QueryType ::= **DMQL2**

An enumeration giving the language in which the query is presented. The only valid value for RETS/1.5 is "DMQL2" which indicates the query language described in Section 7.7

7.4 Optional Request Arguments

7.4.1 Count

The Count argument controls whether the server's response includes a count.

Count ::= **0 | 1 | 2**

If this argument is set to one ("1"), then a record-count is returned in the response in addition to the data. Note that on some servers this will cause the search to take longer since the count must be returned before any records are received. If this entry is set to two ("2") then only a record-count is returned; no data is returned. If this entry is not present or set to zero ("0") there is no record count returned.

Example: Count=2

Instructs the server to return only a count of the records matching the query.

7.4.2 Format

The Format argument selects one of the three supported data return formats for the query response.

Format ::= **COMPACT** | **COMPACT-DECODED** | **STANDARD-XML** | **STANDARD-XML:***dtd-version*

"COMPACT" means a field list <COLUMNS> followed by a delimited set of the data fields. "COMPACT-DECODED" is the same as COMPACT except the data is returned in a fully-decoded (people-readable) format. See Section 13 for more information on the COMPACT formats. "STANDARD-XML" means an XML presentation of the data in the format defined by the RETS Data XML DTD. Servers MUST support all formats. If the format is not specified, the server MUST return STANDARD-XML.

Example: `Format=COMPACT-DECODED`

If the client has requests STANDARD-XML, it may also append a preferred DTD version. The server SHOULD support at least the current version and the prior one.

Example: `Format=STANDARD-XML:1.0`

7.4.3 Limit

The Limit argument requests the server to apply or suspend a limit on the number of records returned in the search.

Limit ::= **NONE** | *1*9DIGIT*

If this entry is set to ("NONE") or is not present, the server SHOULD treat this as a request to suspend enforcement of the standard download limit. The use of "NONE" MAY disable both the <MAXROWS> tag and return-code "20208 Maximum Records Exceeded". Client implementers should be aware that some server implementations might not honor the request to disable the limit; the server operator's business rules take precedence over the request to waive the system download limit.

Alternatively, if the entry is set to a number greater than '0', the server MUST not return more than the specified number of records. If the server did not return all matching records then the <MAXROWS> tag MUST be sent at the end of the data stream.

7.4.4 Offset

The client may specify that a retrieval start at other than the first record in the set of records matching he query by specifying the Offset argument.

Offset ::= *1*9DIGIT*

This argument indicates to the server that it should start sending the data to the client beginning with the record number indicated, with a value of "1" indicating to start with the first record. This can be useful when requesting records in batches, however, client implementers should be aware that data on the server MAY change as they iterate through the batches and it is possible that some records may be missed or added. In other words, the server is not required to maintain a cursor to the data.

7.4.5 Select

By default, the server MUST return all fields accessible to the client. The client may select a subset of those fields by specifying the Select argument.

Select ::= *field**(, *field*)

This parameter is used to set the fields that are returned by the query. If this entry is not present then all allowable fields for the search/class are returned. The server MAY return an error when there are unknown fields in the select list. The server MUST NOT return more fields than are specified in the Select argument when the client requests COMPACT or COMPACT-DECODED data. It MAY return fewer if some of the field names are invalid or if a requested field is unavailable to the user based on security or other restrictions.

7.4.6 Restricted Indicator

In some instances, the server may withhold the values of selected fields on selected records. In this case, the server SHOULD send back a null value, unless the client has specified a RestrictedIndicator.

RestrictedIndicator ::= *1*9ALPHANUM*

This entry indicates to the server that it should set the restriction indicator to the value specified by this tag. The default is that the server returns no restriction indicator.

Example: `RestrictedIndicator = ####`

This would mean that all fields that the user is not allowed to see within a record (e.g. ExpirationDate) are returned with a value of ####.

Note that if the client requests fields that the server would withhold for every record, the server MAY choose to omit the field from the list returned rather than use the RestrictedIndicator for every record.

7.4.7 StandardNames

Queries may use either standard names or system names in the query (Section 7.7). If the client chooses to use standard names, it MUST indicate this using the StandardNames argument.

StandardNames ::= **0** | **1**

If this entry is set to (“0”) or is not present the field names passed in the search query are the SystemNames, as defined in the metadata. If this entry is set to (“1”) then the StandardNames are used for the field names passed in the search query.

7.5 Required Response Arguments

There are no required response arguments.

7.6 Search Response Body Format

The body of the search response has the following format when replying to a request with the format set to "COMPACT" or "COMPACT-DECODED":

```
<RETS 1*SP ReplyCode= quoted-reply-code 1*SP
      ReplyText= quoted-string *SP > CRLF
[ count-tag ]
[ delimiter-tag ]
[ column-tag ]
*( compact-data )
[ max-row-tag ]
[<RETS-STATUS [1*SP ReplyCode= quoted-end-reply-code 1*SP
      ReplyText= quoted-string *SP]/>]
</RETS> CRLF
```

The body of the search response has the following format when replying to a format request of "STANDARD-XML" data:

```
<?xml version="1.0" ?>
[doctype]
<RETS 1*SP ReplyCode= quoted-reply-code 1*SP
      ReplyText= quoted-string *SP >
[ count-tag ]
*( XML-data-record )
[ max-row-tag ]
[<RETS-STATUS [1*SP ReplyCode= quoted-end-reply-code 1*SP
      ReplyText= quoted-string *SP]/>]
</RETS> CRLF
```

```
doctype        ::=  <!DOCTYPE RETS SYSTEM "dtd-version">

dtd-version    ::=  <Name of the RETS DTD used to produce this document>
```

When the client requests the STANDARD-XML representation, it may also specify a DTD version. The server SHOULD be prepared to support at least the current version and the prior version. Data DTD versions are of the form

RETS-*yyyymmdd*.dtd

where *yyyymmdd* is the release date of the DTD.

```
compact-data   ::=  <DATA> field-delimiter*( field-data field-delimiter)
                    </DATA> CRLF
```

If a "COMPACT" or "COMPACT-DECODED" format is specified in the request then a "<DATA>" tag, a delimited list of field-data and a "</DATA>" end tag are returned to the client for each record returned. The field-delimiter is determined by the delimiter-tag.

```
count-tag      ::=  <COUNT 1*SP Records="record-count" 1*SP /> CRLF
```

When the client application specifies that a count should be returned (count-type = "1" | "2") a count-tag MUST be sent by the server in the response. The "<COUNT>" tag MUST be on the first line following the reply-code line. The record-count value indicates the number of records on the server matching the search criteria sent in the search query.

```
column-tag     ::=  <COLUMNS> field-delimiter 1*( field field-delimiter )
                    </COLUMNS> CRLF
```

If a "COMPACT" format is specified in the request then a "<COLUMNS>" tag, including a delimited list of the names of all the fields of data being returned, is sent back in the response. These names are the system-names unless standard-names were used in the query.

The field-delimiter is determined by the delimiter-tag.

```
delimiter-tag  ::=  <DELIMITER value =" field-delimiter "/> CRLF
```

This parameter tells the client which character (OCTET) to use as a delimiter for both the COLUMNS list and the DATA returned. The server MUST send this parameter for

"COMPACT" or "COMPACT-DECODED" formats. The "<DELIMITER>" tag MUST precede column-tag.

max-row-tag ::= **<MAXROWS/>** *CRLF*

A tag that indicates the maximum number of records allowed to be returned by the server has been exceeded, or alternatively, the Limit number passed by the client in the request has been exceeded.

7.7 Query language

The query takes the form indicated below. This is the actual search criteria passed to the server. The server parses this query and generates a server-compatible query based on the parameters passed in the query-list.

7.7.1 Query language BNF

search-condition ::= *query-clause* | (*search-condition or query-clause*)

query-clause ::= *boolean-element* | (*query-clause and boolean-element*)

boolean-element :: = [*not*] *query-element*

query-element ::= *field-criteria* | (**(** search-condition **)**)

or ::= **OR** | **|**

and ::= **AND** | **,**

not ::= **NOT** | **~**

field-criteria ::= **(** *field* **=** *field-content* **)**

field-value ::= *lookup-list* | *string-list* | *range-list* | *period* | *number* | *string-literal*

lookup-list ::= *lookup-or* | *lookup-not* | *lookup-and*

lookup-or ::= **|** *lookup* *(**,** *lookup*)

lookup-not ::= **~** *lookup* *(**,** *lookup*)

lookup-and ::= **+** *lookup* *(**,** *lookup*)

lookup ::= <any legal ALPHANUM value for the field as defined in the metadata>

string-list ::= 1*(*string* *(**,** *string*))

string ::= *string-eq* | *string-start* | *string-contains* | *string-char*

string-eq ::= 1*ALPHANUM

string-start ::= 1*ALPHANUM *****

string-contains ::= ***** 1*ALPHANUM *****

string-char ::= *ALPHANUM 1***?** *ALPHANUM

string-literal ::= **"** 1*(*(PLAINTEXT except **"**) *(2 * **"**) *(PLAINTEXT except **"**)) **"**

range-list	::= 1*(*range* *(, *range*))
range	::= *between* | *greater* | *less*
between	::= (*period* | *number* | *string-eq*) - (*period* | *number* | *string-eq*)
greater	::= (*period* | *number* | *string-eq*) +
less	::= (*period* | *number* | *string-eq*) -
period	::= (*date* | *datetime* | *time*)
number	::= 1*DIGIT ["." *DIGIT]
date	::= (*year* - *month* - day) | **TODAY**
datetime	::= (*year* - *month* - *day* **T** *hour* **:** *minute* **:** *second* [**.** *fraction*]) | **NOW**
time	::= (hour ":" minute ":" second ["." fraction])
fraction	::= 1*3DIGIT
second	::= 2DIGIT
minute	::= 2DIGIT
hour	::= 2DIGIT
day	::= 2DIGIT
month	::= 2DIGIT
year	::= 4DIGIT

7.7.2 Query parameter interpretation

All datetimes submitted in queries MUST be in GMT. All other dates or times are interpreted in host time. The host MUST interpret the token **NOW** as the current date and time, and the token **TODAY** as the current date (with a time of 0000 if the host uses full timestamps).

In processing a literal string, a server MAY substitute a *string-char* expression (**?**s) for the range of characters that contain any non-ALPHANUM not supported by that server.

In processing decimal numbers, where rounding is necessary, a server SHOULD round down for the bottom of ranges or values less than .5 and round up for the tops of ranges or values .5 or greater.

There are three types of field values that can be passed in the query string. They are a *lookup-list*, a *range* and a *string*. A *lookup-list* is a field that may only contain predefined values. “Status” and “Type” typically fall into this category.

A *range* field is of type numeric or date. These fields can be searched based on a range of values. “ListPrice” and “ListDate” fall into this category. All values specified in a <range> field are to be treated as inclusive (e.g. 2+ is the same as 2 or greater, inclusive of 2; 2-3 is the same as 2 to 3, inclusive of 2 and 3; 2- is the same as 2 or less, inclusive of 2).

A *string* field is any other character field not falling into the other two categories. These are usually freeform text fields. An example of this kind of field is “OwnerName”.

Each *field* MUST be a SystemName, as defined in the metadata, unless the StandardName argument is set to "1", in which case the *field*s MUST be StandardNames. All values submitted for lookup-lists must be the Value in compact format, as defined in Section 13.

7.7.3 Sub-queries

This query language provides for a nesting of sub-queries. For example:

```
Query=((AREA=|1,2)|(CITY=ACTON)),(LP=200000+)
```

Example:

```
Query=(ST=|ACT,SOLD),
   (LP=200000-350000),
   (STR=RIVER*),
   (STYLE=RANCH),
   (EXT=+WTRFRNT,DOCK),
   (LDATE=2000-03-01+),
   (REM=*FORECLOSE*),
   (TYPE=~CONDO,TWNHME),
   (OWNER=P?LE)
```

Verbally, this would be interpreted as "return properties with (ST equal ACT or SOLD) and (LP between 200000 and 350000, inclusive) and (STR beginning with RIVER) and (STYLE equal RANCH) and (EXT equal WTRFRNT and DOCK) and (LDATE greater than or equal to 2000-03-01) and (REM containing FORECLOSE) and (TYPE not equal to CONDO and not equal to TWNHME) and (OWNER starting with P and having LE in the 3rd and 4th characters)."

7.8 Reply Codes

Table 7-1 Search Transaction Reply Codes

Reply Code	Meaning
0	Operation successful.
20200	Unknown Query Field The query could not be understood due to an unknown field name.
20201	No Records Found No matching records were found.
20202	Invalid Select The Select statement contains field names that are not recognized by the server.
20203	Miscellaneous Search Error The quoted-string of the body-start-line contains text that MAY be displayed to the user.
20206	Invalid Query Syntax The query could not be understood due to a syntax error.
20207	Unauthorized Query The query could not be executed because it refers to a field to which the supplied login does not grant access.
20208	Maximum Records Exceeded Operation successful, but all of the records have not been returned. This reply code indicates that the maximum records allowed to be returned by the server have been exceeded. Note: reaching/exceeding the "Limit" value in the client request is not a cause for the server to generate this error.
20209	Timeout The request timed out while executing
20210	Too many outstanding queries The user has too many outstanding queries and new queries will not be accepted at this time.
20514	Requested DTD version unavailable. The client has requested the metadata in STANDARD-XML format using a DTD version that the server cannot provide.

SECTION

8

GET TRANSACTION

Gets an arbitrary file from the server or performs an arbitrary action, specified by URI. This is a standard HTTP GET, per RFC 2616. The file to get is passed as part of the Request-URI.

RETS servers need not support the GET transaction to any greater extent than is necessary to implement the functionality of the Action URL (see 4.10, "Capability URL List"). If a RETS server does not intend to include an Action URL in its login responses, it need not support the GET transaction.

8.1 Required Request Arguments

There are no required request arguments.

8.2 Optional Request Arguments

There are no optional request arguments.

8.3 Required Response Arguments

There are no required response arguments.

8.4 Optional Response Arguments

There are no optional response arguments.

8.5 Status Conditions

See the General Status Codes in Section 3.10 for typical Status-Codes.

SECTION

9

CHANGE PASSWORD TRANSACTION

The Change Password transaction provides a means for the user to change their password. The new password is appended to the username and encrypted using the Data Encryption Standard (DES), ANSI X3.92, using a hash of the old password as the key.

9.1 Required Request Arguments

PWD ::= **PWD=** <DES(*Password* **:** *UserName*)>

This is the DES-encrypted UserName and Password. The new Password and the UserName are appended together with a colon (":") between and the resulting string is encrypted using DES in Electronic Code Book (ECB) mode. The DES key is constructed using the procedure in Section 9.6.

9.2 Optional Request Arguments

There are no optional request arguments.

9.3 Required Response Arguments

There are no required response arguments.

9.4 Optional Response Arguments

There are no optional response arguments.

9.5 Reply Codes

Table 9-1 Change Password Reply Codes

Reply Code	Meaning
0	Operation successful.
20140	Insecure password. The password does not meet the site's rules for password security.
20141	Same as Previous Password. The new password is the same as the old one.
20142	The encrypted user name was invalid.

9.6 Encryption Key Construction

The new password is communicated to the host as a string encrypted with the Data Encryption Standard, ANSI X3.92. DES requires a 56-bit key, which is constructed as follows:

1. The old password and username are converted to uppercase and concatenated together.
2. The resulting string is hashed using MD5.
3. The key is taken as the first 56 bits of the resulting hash value.

SECTION

10

UPDATE TRANSACTION

The update transaction is used to modify data on the server. The client transmits information describing the update to perform. The information is then validated by the server. If there are errors in the data, the server returns an error reply. If there are no errors, the record as it was inserted/updated on the server will be returned. The record is returned in the same manner as a record is returned from a search.

Update requests MUST use the POST method (rather than the GET method). This allows the client to transmit characters beyond the HTTP length limit for the GET method.

10.1 Required Request Arguments

The request has the following format:

```
Resource= resource-name
&ClassName= class-name
&Validate= validate-flag
&Type= update-type
&Delimiter= field-delimiter
&Record= field-name = field-value *( field-delimiter field-name =
field-value )
```

resource-name ::= *1*32ALPHANUM*

The name of the resource to be updated, as specified in the metadata. This is the SystemName as defined in Section 11.2.2.

class-name ::= *1*24ALPHANUM*

The name of the class to be updated, as defined in the metadata. This is the ClassName as defined in section 11.3.1.

validate-flag ::= **1** | **0**

If this parameter is set to one ("**1**"), then the record is validated by the host. Any fields with metadata field "Attributes" set to "Autopop" in the metadata (see Section 11.3.4) will have their field values filled in by the server and returned to the client. The record in the server database is not updated. If this entry is set to zero ("**0**") and there are no errors in the record the record is updated on the server.

update-type ::= *1*24 ALPHANUM*

The type of update to perform, as specified by the metadata. This is the UpdateType as defined in Section 11.3.4.

field-name ::= *1*32ALPHANUM*

The name of the field to be updated, as specified in the metadata. This is the SystemName as defined in Section 11.3.2.

field-delimiter ::= *OCTET*

The octet which will separate fields in the record. If this is not specified, an ASCII HT character is assumed.

field-value ::= <varies depending on the field>

The text representation of the field value as defined by the metadata in Section 11.3.2 subject to the business rules.

10.2 Optional Request Arguments

There are no optional request arguments.

10.3 Required Response Arguments

There are no required response arguments.

10.4 Optional Response Arguments

There are no optional response arguments.

10.5 Update Response Body Format

The body of the update response has the following format when there are no errors:

```
<RETS 1*SP ReplyCode= quoted-reply-code 1*SP
           ReplyText= quoted-string *SP > CRLF
transaction-id-tag
[ delimiter-tag ]
column-tag
compact-data
[<RETS-STATUS 1*SP ReplyCode= quoted-end-reply-code 1*SP
           ReplyText= quoted-string *SP/>
</RETS> CRLF
```

The body of the update response has the following format when there are errors:

```
<RETS 1*SP ReplyCode= quoted-reply-code 1*SP
           ReplyText= quoted-string *SP > CRLF
transaction-id-tag
[ delimiter-tag ]
column-tag
compact-data
error-block
</RETS> CRLF

error-block    = <ERRORBLOCK> CRLF
                 1*(<ERRORDATA> field error-num error-offset error-text
                 </ERRORDATA>CRLF)
                 </ERRORBLOCK> CRLF
```

An Error Block is returned when there is a problem with one or more of the fields. The error block contains information about the fields that have errors. It contains the field

name, an error number, some additional text about the error (*error-text*), and where in the field data the error occurred (*error-offset*).

error-num ::= *1*5DIGIT*

This is the host error number. This number along with the *error-text* MAY be displayed to the user when looking at the corresponding field in the client application.

error-offset ::= *1*5DIGIT*

This is the offset into the field data that was sent by the client application to the server. It indicates at what character in the field data the problem was encountered. This number is set to zero ("0") is the offset of the error is unknown.

error-text ::= *64ALPHANUM*

This is the error text generated by the host to assist the user in determining the problem with the field data. This text is associated with the *error-num*.

10.6 Validation

Validation routines are indications of the checks the host system will perform against a field value before it is accepted for storage on the host. Some of these routines require data available only on the host system. However, others are relatively simple and could be performed by any RETS client to prevent invalid field values from being submitted.There are several different types of validation to be performed by the client.

A compliant client is not required to enforce the local validations provided in this section. However, if a client does not enforce the validations then the likelihood of the server rejecting the record is greatly increased.

10.6.1 Lookup

The entry is validated against a list of acceptable values. If the metadata described in Section 11.3.2 specifies the Interpretation as Lookup the only acceptable values for the field are defined in the METADATA-LOOKUP referenced by LookupName. Alternatively, if the metadata specifies a ValidationLookup the only acceptable values for the field are defined in the METADATA-VALIDATION_LOOKUP referenced by the ValidationLookup field.

10.6.2 MultiSelect Lookup

The entry is validated against a list of acceptable values. If the metadata described in Section 11.3.2 specifies the Interpretation as LookupMulti, LookupBitstring or LookupBitmask the only acceptable values for the field are defined in the METADATA-LOOKUP referenced by LookupName. The maximum number of values that can be selected is defined by MaxSelect.

10.6.3 Range

The entry must be between the Minimum and Maximum values specified in the metadata (see Section 11.3.2).

10.6.4 Test Expression

The parameter list contains an expression evaluated by the routine. If the expression is true, the value of the field is acceptable. If the expression is false, the value is rejected. See Section 11.4.9 for more information on Test Expressions.

10.6.5 External

The entry may be validated by searching a server resource. The Resource is defined for searching and the parameter list includes a set of suggested input fields, a set of result fields to display and a set of result fields to populate into the fields of the resource being updated. Information for external validation is provided in Section 11.4.10.

10.7 Reply Codes

Table 10-1 Update Transaction Reply Codes

Reply Code	Meaning
0	Operation successful.
20301	Invalid parameter. Additional information is provided in the error block.
20302	Unable to save record on server.
20303	Miscellaneous Update Error.

The quoted-string of the body-start-line contains text that MAY be displayed to the user.

SECTION

11

METADATA FORMAT

Metadata enables a client that receives data from a compliant server to better format the data for display, and to store it efficiently for future retrieval. While use of the metadata is not necessary to retrieve data for simple display purposes, more sophisticated clients will want to use the metadata to make more intelligent use of the information retrieved. Metadata MUST be supplied by a compliant server.

11.1 Organization and Retrieval

11.1.1 Metadata Organization

Metadata is organized by table/object, with each table having its own unique set of metadata describing the fields available in that table/object. The organization permits access to summary or detailed information about one or more resources (see Figure 11.1, "Metadata Structure").

The client retrieves the metadata by using the GetMetadata Transaction specifying the METADATA table/object(s) of interest as the Type, and the specific instance in the ID (see Section 5). The server supplies the metadata as documents using the formats described in this section. The client MUST accept fields and attributes in the metadata that are not defined in this standard, although it is not required to process those fields in any way.

The client may cache the metadata between sessions. If it does, it MUST record the value of the METADATA-SYSTEM Version attribute from each session in which it caches retrieved metadata, and MUST request new metadata whenever the `MetadataVersion` Login response value changes except when previous versions are permitted by the `MinMetadataVersion` value . The server MUST maintain a single version value for all metadata changes. When changes are made, the version must be increased and that version value and its date MUST be applied to all metadata elements and their parents affected by the change or simultaneous group of changes.

11.1.2 General Rules for Interpretation

In general, metadata keywords defined in this standard such as field names and reserved values are not case-sensitive. However, implementers are urged to adopt the strict-generation/tolerant-acceptance rule and follow the case shown in this standard.

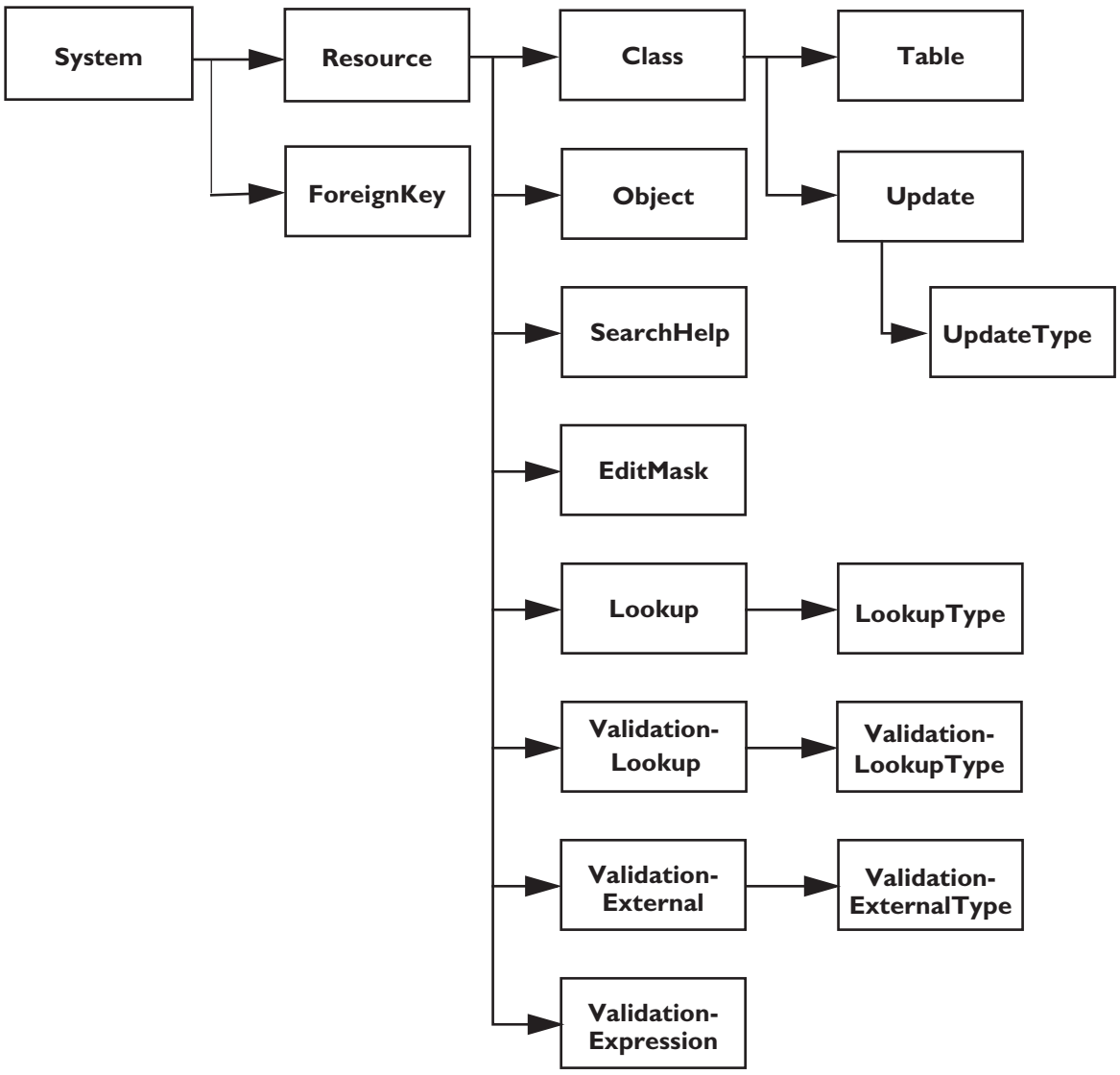

Figure 11.1 Metadata Structure

Servers may choose to extend the content of any metadata table by including additional keywords. Metadata field names for such extensions SHOULD begin with the letter "X" followed by a hyphen, followed by an implementation-defined token in order to insure compatibility with future versions of the standard.

Clients MUST ignore any metadata fields which they do not understand.

11.1.3 Metadata Retrieval Hierarchy

The ID argument in the GetMetadata transaction reflects the metadata hierarchy as shown in Figure 11.1. For any metadata element, the ID argument is a list of the names of the parent elements for the desired element, separated by colons. For example, to retrieve the EditMask table for a given named Resource, the argument is simply the ResourceID:

```
Type: METADATA-EDITMASK
ID: Property
```

where `Property` is the ID of one of the Resources listed in the Metadata-Resource table.

Since Tables are children of Classes, which are in turn children of Properties, the ID parameter contains both parents:

```
Type: METADATA-TABLE
ID: Property : Res
```

where `Res` is a class listed in the Metadata-Class table under the resource `Property`.

11.1.4 Hierarchical Metadata in COMPACT Format

Metadata may be retrieved in either COMPACT or XML format (see Section 12). Compact metadata is intended as a lower-bandwidth alternative to XML retrieval. It is not intrinsically hierarchical; sections are not nested within one another but are entirely self-contained as shown in the examples. If the client requests a hierarchical retrieval using a wild card for one of the levels, the tables at that and lower levels are sent sequentially. The only requirement for order is that there be no forward references within the metadata; that is, higher-level tables such as `METADATA-RESOURCE` MUST be sent before lower-level tables such as `METADATA-LOOKUP`.

11.2 System-Level Metadata

Clients can determine the number and type of searchable and updateable entities by referencing the Resources. A server MUST advertise its resources. It MAY advertise all of its available resources or MAY restrict the advertised list by logon or other criteria. A server's advertisement of a resource does not require that the server be able to accommodate any arbitrary search for that user; the server MAY restrict access to resources that it advertises. If the server supports multimedia objects then it MUST advertise the supported types.

All resources that can be searched or updated are defined in the metadata described in this section. There are three parts to the metadata. The first part provides system information and describes the available resources, the second part describes the class specific metadata for a resource, and the third part describes the shared metadata for a resource.

11.2.1 System

The System metadata starts with a <METADATA-SYSTEM> tag with Version and Date attributes. This tag is followed by a <SYSTEM> section, which contains the system identification information. An optional <COMMENTS> section completes the System metadata. The System metadata has the following format:

```
<METADATA-SYSTEM· Version="system-version" · Date="system-date" > ↲
<SYSTEM· SystemID="code-name" · SystemDescription="long-name" /> ↲
[ <COMMENTS> ↲
 *( comment ↲)
 </COMMENTS> ↲ ]
</METADATA-SYSTEM> ↲
```

System Version

system-version ::= *1*2DIGITS . 1*2DIGITS . 1*5DIGITS*

This is the version of the document. The convention used is a "<major>.<minor>.<release>" numbering scheme. Every time any metadata element changes the version number MUST be increased.

System Date

system-date ::= *DATE*

The System Date is latest change date of any System metadata.

System Description

code-name ::= *1*10ALPHANUM*

long-name ::= *1*64PLAINTEXT*

comments ::= *TEXT*

An example Version section follows:

GetMetadata request:

```
Type: METADATA-SYSTEM
ID: 0
```

Compact reply:

```
<METADATA-SYSTEM Version="1.00.000" Date="Sat, 20 Mar 2002 12:03:38 GMT">
<SYSTEM SystemID= "NTREIS" SystemDescription= "North Texas Real Estate
Information System" />
<COMMENTS>
This is a comment line
</COMMENTS>
</METADATA-SYSTEM>
```

11.2.2 Resources

RETS does not require that any particular type of data be made available by a server. However, a server MUST use a standard well-known name under which to make its data available if a suitable well-known name is defined in the standard. Table 11-1 contains the list of well-known resource names.

Table 11-1 Well-Known Resource Names

Resource Name	Purpose
ActiveAgent	A resource that contains information about active agents. These are agents that are currently authorized to access the server (paid-up, not retired, etc.)
Agent	A resource that contains information about agents.
History	A resource that contains information about the accumulated changes to each listing.
Office	A resource that contains information about broker offices.
OpenHouse	A resource that contains information about open-house activities.

Table 11-1 Well-Known Resource Names (continued)

Resource Name	Purpose
Property	A resource that contains information about listed properties. Information in this resource is described by Real Estate Transaction XML DTD in addition to appropriate metadata.
Prospect	A resource that contains information about sales or listing prospects.
Tax	A resource that contains tax assessor information.
Tour	A resource that contains information about tour activities.

Resource Metadata Content

The Resource metadata starts with a <METADATA-RESOURCE> tag with Version, and Date attributes. This is followed by a <COLUMNS> section that contains the name of the fields as defined in Table 11-2 followed by the <DATA> section that contains the actual field information. The Resource metadata has the following format:

```
<METADATA-RESOURCE · Version="resource-version" ·Date="resource-date">↲
<COLUMNS>→resource-field *(→resource-field)→</COLUMNS>↲
*(<DATA>→resource-data *(→resource-data)→</DATA>↲)
</METADATA-RESOURCE>↲
```

resource-version::= 1*2DIGITS . 1*2DIGITS . 1*5DIGITS

This is the version of the Resource metadata. It is the highest version number of any of the contained metadata elements. The convention used is a "<major>.<minor>.<release>" numbering scheme. Every time any contained metadata element changes the version number MUST be increased.

resource-date ::= *DATE*

The latest change date of any contained metadata.

resource-field = <Field Name from Table 11-2>

resource-data ::= <valid value as defined in Table 11-2>

An example Resource section follows:

GetMetadata request:

```
Type: METADATA-RESOURCE
ID: 0
```

Compact reply:

```
<METADATA-RESOURCE Version="1.00.000"
   Date="Sat, 20 Mar 2002 12:03:38 GMT" >
<COLUMNS>→ResourceID→StandardName→VisibleName→Description→
   ClassCount→KeyField→ClassVersion→ClassDate→ObjectVersion→
   ObjectDate→SearchHelpVersion→SearchHelpDate→EditMaskVersion→
   EditMaskDate →LookupVersion→LookupDate→UpdateHelpVersion→
   UpdateHelpDate →ValidationExpressionVersion→
   ValidationExpressionDate→ValidationLookupVersion →
   ValidationLookupDate→ValidationExternalVersion→
   ValidationExternalDate→</COLUMNS>
<DATA>→Agent→Agent→ Agent→Agent Table→1→ Agentid→1.00.000→
   Sat, 20 Mar 2002 12:03:38 GMT→→→→→→→→→→→→→→→→→→</DATA>
<DATA>→Property→Property→Property→Property Tables→5→
   LN→1.00.000→ Sat, 20 Mar 2002 12:03:38 GMT→1.00.000→
   Sat, 20 Mar 2002 12:03:38 GMT→1.00.000→
```

```
    Sat, 20 Mar 2002 12:03:38 GMT →1.00.000→
    Sat, 20 Mar 2002 12:03:38 GMT →1.00.000→
    Sat, 20 Mar 2002 12:03:38 GMT →1.00.000→
    Sat, 20 Mar 2002 12:03:38 GMT →1.00.000→
    Sat, 20 Mar 2002 12:03:38 GMT →1.00.000→
    Sat, 20 Mar 2002 12:03:38 GMT →1.00.000→
    Sat, 20 Mar 2002 12:03:38 GMT →</DATA>
<DATA>→Tax→Tax→Tax→Multimedia objects→2→0→ PID→1.00.000→
    Sat, 20 Mar 2002 12:03:38 GMT→→→→→→→→→→→→→→→→→→→</DATA>
</METADATA-RESOURCE>
```

Table 11-2 Metadata: Resource Description Fields

Field Name	Content Type	Description
ResourceID	*1*32ALPHANUM*	The name which acts as a unique ID for this resource.
StandardName	*1*32ALPHANUM*	The name of the resource. This must be a well-known name if applicable.
VisibleName	*1*32PLAINTEXT*	The user-visible name of the resource.
Description	*1*64PLAINTEXT*	A user-visible description of the resource.
KeyField	*1*32ALPHANUM*	The SystemName (see 11.3.2) of the field that provides a unique ResourceKey for each element in this resource. All classes within a resource must use the same KeyField.
ClassCount	Numeric	The number of classes in this resource. There MUST be ClassCount METADATA_CLASS descriptions for the resource. There MUST be at least one Class for each Resource.
ClassVersion	*1*2DIGITS . 1*2DIGITS . 1*5DIGITS*	The latest version of the Class metadata for this Resource. The convention used is a "<major>.<minor>.<release>" numbering scheme. Clients MAY rely on this data for cache management.
ClassDate	*DATE*	The date on which the Class metadata for this Resource was last changed. Clients MAY rely on this date for cache management.
ObjectVersion	*1*2DIGITS . 1*2DIGITS . 1*5DIGITS*	The version of the Object metadata for this Resource. The convention used is a "<major>.<minor>.<release>" numbering scheme. Clients MAY rely on this data for cache management. A blank version indicates no Object metadata is available for this Resource.
ObjectDate	*DATE*	The date on which the Object metadata for this Resource was last changed. Clients MAY rely on this date for cache management. A blank date indicates no Object metadata is available for this Resource.
SearchHelpVersion	*1*2DIGITS . 1*2DIGITS . 1*5DIGITS*	The version of the SearchHelp metadata for this Resource. The convention used is a "<major>.<minor>.<release>" numbering scheme. Clients MAY rely on this data for cache management. A blank version indicates no SearchHelp is available for this Resource.

Table 11-2 Metadata: Resource Description Fields (continued)

Field Name	Content Type	Description
SearchHelpDate	*DATE*	The date on which the SearchHelp metadata for this Resource was last changed. Clients MAY rely on this date for cache management. A blank date indicates no SearchHelp is available for this Resource.
EditMaskVersion	1*1*2DIGITS . *1*2DIGITS . *1*5DIGITS	The version of the EditMask metadata for this Resource. The convention used is a "<major>.<minor>.<release>" numbering scheme. Clients MAY rely on this data for cache management. A blank version indicates no EditMask is available for this Resource.
EditMaskDate	*DATE*	The date on which the EditMask metadata for this Resource was last changed. Clients MAY rely on this date for cache management. A blank date indicates no EditMask is available for this Resource.
LookupVersion	*1*2DIGITS . *1*2DIGITS . *1*5DIGITS	The version of the Lookup metadata for this Resource. The convention used is a "<major>.<minor>.<release>" numbering scheme. Clients MAY rely on this data for cache management. A blank version indicates no Lookup is available for this Resource.
LookupDate	*DATE*	The date on which the Lookup metadata for this Resource was last changed. Clients MAY rely on this date for cache management. A blank date indicates no Lookup is available for this Resource.
UpdateHelpVersion	*1*2DIGITS . *1*2DIGITS . *1*5DIGITS	The version of the UpdateHelp metadata for this Resource. The convention used is a "<major>.<minor>.<release>" numbering scheme. Clients MAY rely on this data for cache management. A blank version indicates no UpdateHelp is available for this Resource.
UpdateHelpDate	*DATE*	The date on which the UpdateHelp metadata for this Resource was last changed. Clients MAY rely on this date for cache management. A blank date indicates no UpdateHelp is available for this Resource.
Validation-ExpressionVersion	*1*2DIGITS . *1*2DIGITS . *1*5DIGITS	The version of the ValidationExpression metadata for this Resource. The convention used is a "<major>.<minor>.<release>" numbering scheme. Clients MAY rely on this data for cache management. A blank version indicates no ValidationExpression is available for this Resource.
Validation-ExpressionDate	*DATE*	The date on which the ValidationExpression metadata for this Resource was last changed. Clients MAY rely on this date for cache management. A blank date indicates no ValidationExpression is available for this Resource.

Table 11-2 Metadata: Resource Description Fields (continued)

Field Name	Content Type	Description
ValidationLookup-Version	*1*2DIGITS* . *1*2DIGITS* . *1*5DIGITS*	The version of the ValidationLookup metadata for this Resource. The convention used is a "<major>.<minor>.<release>" numbering scheme. Clients MAY rely on this data for cache management. A blank version indicates no ValidationLookup is available for this Resource.
ValidationLookup-Date	*DATE*	The date on which the ValidationLookup metadata for this Resource was last changed. Clients MAY rely on this date for cache management. A blank date indicates no ValidationLookup is available for this Resource.
ValidationExternal-Version	*1*2DIGITS* . *1*2DIGITS* . *1*5DIGITS*	The version of the ValidationExternal metadata for this Resource. The convention used is a "<major>.<minor>.<release>" numbering scheme. Clients MAY rely on this data for cache management. A blank version indicates no ValidationExternal is available for this Resource.
ValidationExternal-Date	*DATE*	The date on which the ValidationExternal metadata for this Resource was last changed. Clients MAY rely on this date for cache management. A blank date indicates no ValidationExternal is available for this Resource.

11.2.3 Metadata Format for Foreign Keys

The ForeignKeys metadata table allows a server to advertise relationships among its offered resources. A RETS client MAY use this information to provide a richer display of related information. The ForeignKeys metadata consists of tuples containing a parent resource type, a child resource type, and the foreign keys used to traverse the relation.

The ForeignKey metadata has the following format:

```
<METADATA-FOREIGNKEYS · Version="foreignkeys-version" ·
        Date="foreignkeys-date">↲
<COLUMNS>→foreignkeys-field *(→foreignkeys-field)→</COLUMNS>↲
*(<DATA>→foreignkeys-data *(→foreignkeys-data)→</DATA>↲)
</METADATA-FOREIGNKEYS>↲
```

ForeignKeys Metadata Content

The compact-format ForeignKeys metadata begins with a <METADATA-FOREIGNKEYS> element with Version and Date attributes. This is followed by a <COLUMNS> tag containing the names of the supplied columns of the ForeignKeys metadata, and this is in turn followed by a <COLUMNS> tag containing the rows of the ForeignKeys metadata table, one per line.

foreignkeys-version::=*1*2DIGITS* . *1*2DIGITS* . *1*5DIGITS*

This is the version of the ForeignKeys metadata. The convention used is a "*<major>*.*<minor>*.*<release>*" numbering scheme. Every time any contained metadata element changes the version number MUST be increased.

foreignkeys-date::= *DATE*

The latest change date of any contained metadata.

foreignkeys-field= <Field Name from Table 11-2>

foreignkeys-data::= <valid value as defined in Table 11-2>

An example ForeignKeys section follows:

GetMetadata request:

```
Type: METADATA-FOREIGNKEYS
ID: 0
```

Compact Reply:

```
<METADATA-FOREIGNKEYS Version="1.00.000000"
           Date="Wed, 23 Jan 2002 12:37:38 GMT">
<COLUMNS>PARENT_RESOURCE_ID→PARENT_CLASS_ID→PARENT_SYSTEMNAME→
CHILD_RESOURCE_ID→CHILD_CLASS_ID→CHILD_SYSTEMNAME→</COLUMNS>
<DATA>→Property→RES→MLSNUM→TAX→TAX→MLSNUM→</DATA>
<DATA>→Property→RES→MLSNUM→History→History→MLSNUM→</DATA>
<DATA>→Property→RES→MLSNUM→OpenHouse→OpenHouse→MLSNUM→</DATA>
<DATA>→Property→RES→ListingAgentID→Agent→Agent→AgentID→</DATA>
<DATA>→Property→RES→COListingAgentID→Agent→Agent→AgentID→</DATA>
<DATA>→Property→RES→SellingAgentID→Agent→Agent→AgentID→</DATA>
<DATA>→Property→RES→COSellingAgentIDvAgent→Agent→AgentID→</DATA>
<DATA>→Property→RES→ListingOfficeID→Office→Office→OfficeID→</DATA>
<DATA>→Property→RES→SellingOfficeID→Office→Office→OfficeID→</DATA>
</METADATA-FOREIGNKEYS>
```

Table 11-3 Metadata Content: Foreign Keys

Metadata Field	Content Type	Description
ForeignKeyID	*1*32ALPHANUM*	A Unique ID that represents the foreign key combination.
ParentResourceID	*1*32ALPHANUM*	The ResourceID (Table 11-2) of the resource for which this field functions as a foreign key . The name given MUST appear in the METADATA-RESOURCE table..
ParentClassID	*1*32ALPHANUM*	The name of the resource class for which this field functions as a foreign key. This name MUST appear in the RESOURCE-CLASS table for the given ParentResourceID.
ParentSystemName	*1*32ALPHANUM*	The SystemName of the field in the given resource class that should be searched for the value given in the this field. This name must appear as a SystemName in the META-DATA-TABLE section of the metadata for the Parent-ClassID, and the named item must have its Searchable attribute set to TRUE.
ChildResourceID	*1*32ALPHANUM*	The ResourceID (Table 11-2) of the resource for which this field functions as a foreign key . The name given MUST appear in the METADATA-RESOURCE table.
ChildClassID	*1*32ALPHANUM*	The name of the resource class for which this field functions as a foreign key. This name MUST appear in the RESOURCE-CLASS table for the given Child_Resource_ID.
ChildSystemName	*1*32ALPHANUM*	The SystemName of the field in the given resource class that should be searched for the value given in this field. This name must appear as a SystemName in the META-DATA-TABLE section of the metadata for the Child-ClassID, and the named item must have its Searchable attribute set to TRUE.

The nesting of foreign keys MUST be such that recursive searches are NOT REQUIRED to obtain data for well-known fields as defined in the RETS DTD. However, nesting of foreign keys is allowed except in these cases.

11.3 Metadata Format for Class Elements

11.3.1 Class

A given data resource may house multiple classes of entries that can be searched or updated separately. The metadata for a resource supporting searchable classes MUST contain a class description for each class supported.

The Class metadata starts with a <METADATA-CLASS> tag with Resource, Version, and Date attributes. This is followed by a <COLUMNS> section, which contains the name of the fields as defined in Table 11-4 followed by the <DATA> section, which contains the actual field information. The Class metadata has the following format:

```
<METADATA-CLASS · Resource="resource-id" · Version="class-version" ·
   Date="class-date"> ↲
<COLUMNS>→class-field *(→class-field)→</COLUMNS> ↲
*(<DATA>→class-data *(→class-data)→</DATA> ↲)
</METADATA-CLASS> ↲
```

resource-id ::= *1*32ALPHANUM*

This value MUST be a ResourceID found in the Resource metadata. It is the Resource to which the Classes belong.

class-version ::= *1*2DIGITS . 1*2DIGITS . 1*5DIGITS*

This is the version of the Class metadata. It is the highest version number of any of the contained metadata elements. The convention used is a "<major>.<minor>.<release>" numbering scheme. Every time any contained metadata element changes the version number MUST be increased.

class-date ::= *DATE*

The latest change date of any of the contained metadata.

class-field ::= <Field Name from Table 11-4>

class-data ::= <valid value as defined in Table 11-4>

An example Resource Class section follows:

GetMetadata request:

```
Type: METADATA-CLASS
ID: 0
```

Compact reply:

```
<METADATA-CLASS Resource="Property" Version="1.00.000"
   Date="Sat, 20 Mar 2002 12:03:38 GMT">
<COLUMNS>→ClassName→VisibleName→StandardName→Description→
   TableVersion→TableDate→UpdateVersion →UpdateDate →</COLUMNS>
<DATA>→RES→Single Family→Residential→
   Single Family Residential→1.00.000→
   Sat, 20 Mar 2002 12:03:38 GMT→1.00.000→
   Sat, 20 Mar 2002 12:03:38 GMT→</DATA>
<DATA>→CON→Condos→CommonInterest→Condos→1.00.000→
```

```
    Sat, 20 Mar 2002 12:03:38 GMT→1.00.000→
    Sat, 20 Mar 2002 12:03:38 GMT→</DATA>
<DATA>→MUL→Multi Family→MultiFamily→
    Multi Family Residential→1.00.000→
    Sat, 20 Mar 2002 12:03:38 GMT→1.00.000→
    Sat, 20 Mar 2002 12:03:38 GMT→</DATA>
<DATA>→MOB→Mobile Home→ResidentialProperty→
    Mobile Homes→1.00.000→Sat, 20 Mar 2002 12:03:38 GMT→
    1.00.000→Sat, 20 Mar 2002 12:03:38 GMT→</DATA>
<DATA>→LND→Lots and Land→Lots and Land→Lots and Land→
    1.00.000→Sat, 20 Mar 2002 12:03:38 GMT→1.00.000→
    Sat, 20 Mar 2002 12:03:38 GMT→</DATA>
</METADATA-CLASS>
<METADATA-CLASS Resource="Agent" Version="1.00.000"
    Date="Sat, 20 Mar 2002 12:03:38 GMT" />
<COLUMNS>→ClassName→VisibleName→StandardName→Description→
    TableVersion→TableDate→UpdateVersion →UpdateDate →</COLUMNS>
<DATA>→Agent→Agent→Agent→All Agents→1.00.000→
    Sat, 20 Mar 2002 12:03:38 GMT→→→</DATA>
</METADATA-CLASS>
```

Table 11-4 Metadata Content: Resource Class

Metadata Field	Content Type	Description
ClassName	*1*32ALPHANUM*	The name which acts as a unique ID for the class.
StandardName	**Residential-Property** **LotsAndLand** **CommonInterest** **MultiFamily**	The XML standard name. This is the name from the Real Estate Transaction XML DTD.
VisibleName	*1*32PLAINTEXT*	The user-visible name of the class.
Description	*1*64PLAINTEXT*	A user-visible description of the class.
TableVersion	*1*2DIGITS* . *1*2DIGITS* . *1*5DIGITS*	The version of the Table metadata that describes this Class. The convention used is a "<major>.<minor>.<release>" numbering scheme. Clients MAY rely on this data for cache management.
TableDate	*DATE*	The date on which the Table metadata for this Class was last changed. Clients MAY rely on this date for cache management.
UpdateVersion	*1*2DIGITS* . *1*2DIGITS* . *1*5DIGITS*	The latest version of any of the Update metadata for this Class. The convention used is a “<major>.<minor>.<release>” numbering scheme. Clients MAY rely on this data for cache management. A blank version indicates no Update is available for this Class.
UpdateDate	*DATE*	The date on which the any of the Update metadata for this Class was last changed. Clients MAY rely on this date for cache management. A blank date indicates no Update is available for this Class.

All tables that can be searched are defined in a document with the format defined in this section. There are three parts to this section. The first part describes the searchable tables, the second part describes the lookups referenced within the table section, and the third describes the help text associated with searches and edit masks associated with updates.

11.3.2 Table

The Table metadata starts with a <METADATA-TABLE> tag with Resource, Class, Version, and Date attributes. This is followed by a <COLUMNS> section, which contains

the name of the fields as defined in Table 11-5, followed by the <DATA> section, which contains the actual field information. The Table metadata has the following format:

```
<METADATA-TABLE SP Resource="resource-id" SP Class="class-id" SP
Version="table-version" SP Date="table-date"> ↲
<COLUMNS>→table-field *(→table-field)→</COLUMNS> ↲
*(<DATA>→table-data *(→table-data)→</DATA> ↲)
</METADATA-TABLE> ↲
```

resource-id ::= *1*32ALPHANUM*

This value MUST be a ResourceID found in the Resource metadata. It is the Resource to which the Classes belong.

class-id ::= *1*32ALPHANUM*

This value MUST be a ClassName found in the Class metadata for this Resource. It is the Class that this Table describes.

table-version ::= *1*2DIGITS . 1*2DIGITS . 1*5DIGITS*

This is the version number of this Table metadata. The convention used is a "<major>.<minor>.<release>" numbering scheme. Every time this Table metadata changes the version number should be increased.

table-date ::= *DATE*

The latest change date of this Table metadata.

table-field ::= <Field Name from Table 11-5>

table-data ::= <valid value as defined in Table 11-5>

An example Table section follows:

GetMetadata request:

```
Type: METADATA-TABLE
ID: Property: RES
```

Compact reply:

```
<METADATA-TABLE Resource="Property"  Class="RES" Version="1.00.000"
    Date= "Sat, 20 Mar 2002 12:03:38 GMT" >
<COLUMNS>→SystemName→StandardName→LongName→DBName→ShortName→
    Maximumlength→DataType→Precision→Searchable→Interpretation→
    Alignment→UseSeparator→EditMaskID→LookupName→MaxSelect→Units→
    Index→Minimum→Maximum→Default→Required→SearchHelpID→</COLUMNS>
<DATA>→LN→ListID→Listing ID→LN→ListID→8→Int→0→1→
    Number→Left→0→→→→→1→→→1→→→</DATA>
<DATA>→PTYP→PropType→Property Type→PT→Prop Type→
    2→Int→0→1→Number→Left→0→→→→→→→→→→→→</DATA>
<DATA>→LP→ListPrice→List Price→LP→Lst Pr→8→Int→0→1→
    Currency→Right→1→→→→→→14→→→2→→→</DATA>
<DATA>→OWN→Owner→Owner Name→OWN→Own Name→20→Character→
    0→0→→Left→0→→→→→→→→→→→→→</DATA>
<DATA>→VEW→View→View→VEW→View→10→Long→0→1→LookupBitmask→Left→
    0→→VEW→1→→→→→→→→→→</DATA>
<DATA>→EF→ExtFeat→Features→EF→Ext Feat→10→Character→0→1→
    LookupMulti→Left→0→→EFT→2→→→→→→→→→</DATA>
<DATA>→SD→SchDist→School District→SD→SchDist→10→Character→
    0→1→Lookup→Left→0→→SD→→→→→→→→→→→</DATA>
<DATA>→AR→MLSArea→MLS Area→AR→Area→4→Int→0→1→Lookup→Left→
```

```
    0→→AR→→→30→→→3→1→→</DATA>
</METADATA-TABLE>
```

The following table lists the minimum acceptable content for server-supplied metadata used in describing a table.

Table 11-5 Metadata Content - Tables

Field Name	Content Type	Description
SystemName	1*32ALPHANUM	The name of the field as it is known to the native server. The system name SHOULD be unique within the Table; client behavior when the Table metadata contains more than one row with a given SystemName is undefined.
StandardName	Alphanumeric	The name of the field as it is known in the Real Estate Transaction XML DTD.
LongName	1*32TEXT	The name of the field as it is known to the user. This is a localizable, human-readable string. Use of this field is implementation-defined; it is expected that clients will use this value as a title for this datum when it appears on a report.
DBName	*1*10ALPHANUM*	A short name that can be used as a database field name. This name may not start with a number nor can it be an ANSI-SQL92 reserved word.
ShortName	*1*24TEXT*	An abbreviated field name that is also localizable and human-readable. Use of this field is implementation-defined. It is expected that clients will use this field in human-interface elements such as pick lists.
MaximumLength	Numeric	The maximum length of the field, in characters. For numeric fields (small, int, long and decimal) this is the display length rather than the storage length, and includes all formatting such as the sign, decimal point, commas or other insertion edits.
DataType	**Boolean**	A truth-value, stored as 1 for true and 0 for false.
	Character	An arbitrary sequence of printable characters.
	Date	A date, in YYYY-MM-DD format.
	DateTime	A timestamp, in YYYY-MM-DD Thh:mm:ss[.sss] format.
	Time	A time, stored in hh:mm:ss[.sss] format.
	Tiny	A signed numeric value that can be stored in no more than 8 bits.
	Small	A signed numeric value that can be stored in no more than 16 bits.
	Int	A signed numeric value that can be stored in no more than 32 bits.
	Long	A signed numeric value that can be stored in no more than 64 bits.
	Decimal	A decimal value that contains a decimal point (see Precision).
Precision	Numeric	The number of digits to the right of the decimal point when formatted.
Searchable	Boolean	A truth-value which indicates that the field is searchable.

Table 11-5 Metadata Content - Tables (continued)

Field Name	Content Type	Description
Interpretation	**Number**	An arbitrary number.
	Currency	A number representing a currency value.
	Lookup	A value that should be looked up in the Lookup Table. This is a single selection type lookup (e.g. STATUS).
	LookupMulti	A value that should be looked up in the Lookup Table. This is a multiple-selection type lookup (e.g. FEATURES) where the character strings representing each selection are separated by commas.
	LookupBitstring	A value that should be looked up in the Lookup Table. This is a multiple-selection lookup that is stored as a bit string. The bit string is represented as a character string containing only the characters 0 and 1. The leftmost character represents the least-significant bit. The lookup value of the bitstring element is the ordinal position of each bit with the rightmost bit designated as bit 0.
	LookupBitmask	A value that should be looked up in the Lookup Table. This is a multiple-selection type lookup that is stored as a bitmask field. Fields of this type are limited to 31 choices.(e.g. VIEW). When converted to binary, each bit represents one of the possible choices. The choices are from lsb to msb. Lookup values are the numeric equivalent of each bit's binary value (i.e., the low order bit represents the first lookup and the high order bit represents the last lookup choice). $2^{(value-1)}$ is added to the total choice when querying for its applicability.
Alignment	**Left**	The value MAY be displayed left aligned.
	Right	The value MAY be displayed right aligned.
	Center	The value MAY be centered in its field when displayed.
	Justify	The value MAY be justified within its field when displayed.
UseSeparator	Boolean	A truth-value which indicates that the numeric value MAY be displayed with a thousands separator.
EditMaskID	*1*32ALPHANUM* Multiple masks are separated by commas	The name of the entry in the METADATA-EDITMASK table (see 11.4.5, "Edit Mask").
LookupName	*1*32ALPHANUM*	The name of the METADATA-LOOKUP containing the lookup data for this field (see Section 11.4.2). Required if Interpretation is Lookup, LookupMulti, LookupBitstring or LookupBitmask.
MaxSelect	Numeric	This field is required if Interpretation is LookupMulti, LookupBitstring or LookupBitmask. This value indicates the maximum number of entries that may be selected in the lookup.
Units	(**Feet** | **Meters** | **SqFt** | **SqMeters** | **Acres** | **Hectares**)	Unit of measure.
Index	Numeric	An indicator that specifies this field is part of an index. The client MAY use this informationto help the user create faster queries.
Minimum	Numeric	The minimum value that may be stored in a field (applies to numeric fields only).
Maximum	Numeric	The maximum value that may be stored in a field (applies to numeric fields only).

Table 11-5 Metadata Content - Tables (continued)

Field Name	Content Type	Description
Default	Serial	The order that fields should appear in a default one-line search result. Fields that should not appear in the default one-line format should have a value of 0, Fields that should never be visible to the user should have a value of –1.
Required	Numeric	A non-zero value indicates the field is required when searching. This value should be sequential starting with one. If multiple fields share the same value, then one of the fields with the same value is required. (e.g. City = 1 & ZipCode = 1 implies that the user is required to include either City or ZipCode in their query).
SearchHelpID	*1*32ALPHANUM*	The name of the entry in the METADATA-SEARCH_HELP table (see Section 11.4.4).
Unique	Boolean	A truth-value which indicates that this field is a unique identifier for the record in which it appears.

Date formats are based on ISO 8601 [7]

11.3.3 Update

A given data resource may house multiple classes of entries that can be updated separately. The metadata for a resource supporting updateable classes MUST contain a Class Table description for each class supported.

The Update Resource metadata starts with a <METADATA-UPDATE> tag with Resource, Class, Version, and Date attributes. This is followed by a <COLUMNS> section, which contains the names of the fields as defined in Table 11-6, “Metadata Content – Update” followed by the <DATA> section, which contains the actual field information. The Update metadata has the following format:

```
<METADATA-UPDATE · Resource="resource-id" · Class="class-id" ·
    Version="update-version" · Date="update-date"> ↲
<COLUMNS>→update-field *(→update-field)→</COLUMNS> ↲
*(<DATA>→update-data *(→update-data)→</DATA> ↲)
</METADATA-UPDATE> ↲
```

resource-id ::= *1*32ALPHANUM*

This value MUST be a ResourceID found in the Resource metadata. It is the Resource to which the Classes belong.

class-id ::= *1*32ALPHANUM*

This value MUST be a ClassName found in the Class metadata for this Resource. It is the Class to which the Updates apply.

update-version ::= *1*2DIGITS . 1*2DIGITS . 1*5DIGITS*

This is the version number of the Update metadata. This is the highest version number of any of the contained metadata (Update Types). The convention used is a “<major>.<minor>.<release>” numbering scheme. Every time any contained element changes the version number MUST be increased.

update-date ::= *DATE*

The latest change date of any contained metadata element.

update-field ::= <Field Name from Table 11-6>

update-data ::= <valid value as defined in Table 11-6>

An example Update Resource section follows:

GetMetadata request:

```
Type:METADATA_UPDATE
ID: Property: RES
```

Compact reply:

```
<METADATA-UPDATE Resource="Property" Class="RES" Version="1.00.000"
    Date= "Sat, 20 Mar 2002 12:03:38 GMT" >
<COLUMNS> →UpdateName→Description→KeyField→Version→Date→</COLUMNS>
<DATA>→Add→Add a new Residential Listing→→1.00.000→
    Sat, 20 Mar 2002 12:03:38 GMT→</DATA>
<DATA>→Change→Change a Residential Listing→ListNumber→1.00.000→
    Sat, 20 Mar 2002 12:03:38 GMT→</DATA>
<DATA>→BOM→Put a Residential Listing Back on Market →ListNumber→
    1.00.000→Sat, 20 Mar 2002 12:03:38 GMT→</DATA>
</METADATA-UPDATE>
```

Table 11-6 Metadata Content – Update

Metadata Field	Content Type	Description
UpdateName	*1*24ALPHANUM*	This identifies the nature of the update, such as "add" or "modify". Some update types, such as changes to a property record (e.g. "Sell", "Back on Market"), will imply a set of business rules specific to the server. However, where possible, the following standard type names should be used: **Update Name** — **Function** **Add** — Add a new record **Clone** — Create a new record by copying an old one **Change** — Change an existing record **Delete** — Delete an existing record
Description	*1*64PLAINTEXT*	A user visible description of the Update Type.
KeyField	*1*32ALPHANUM*	The SystemName (see Section 11.3.2) of the field that must be used to retrieve an existing record for the update.
Version	*1*2DIGITS* . *1*2DIGITS* . *1*5DIGITS*	The latest version of this Update Type metadata. The convention used is a "<major>.<minor>.<release>" numbering scheme. Clients MAY rely on this data for cache management.
Date	*DATE*	The date on which any of the content of this Update Type was last changed. Clients MAY rely on this date for cache management.

11.3.4 Update Type

A given resource may house multiple classes of entries that can be updated separately. Each of these classes may have different types of updates that can be performed. There might be different test expressions or sequences. This section describes how each of those are specified.

The Update Type metadata starts with a <METADATA-UPDATE_TYPE> tag with Resource, Class, Type, Version, and Date Attributes. This is followed by a <COLUMNS> section, which contains the name of the fields as defined in Table 11-7 followed by the

<DATA> section, which contains the actual field information. The metadata has the following format:

```
<METADATA-UPDATE_TYPE · Resource="resource-id" · Class="class-id" ·
   Version="update-type-version" · Date="update-type-date"> ↲
<COLUMNS>→update-type-field *(→update-type-field)→</COLUMNS> ↲
*(<DATA>→update-type-data *(→update-type-data)→</DATA> ↲)
</METADATA-UPDATE_TYPE> ↲
```

resource-id ::= *1*32ALPHANUM*

This value MUST be a ResourceID found in the Resource metadata. It is the Resource to which the Update Type belongs.

class-id ::= *1*32ALPHANUM*

This value MUST be a ClassName found in the Class metadata for this Resource. It is the Class to which this Update Type applies.

update-type ::= *1*32ALPHANUM*

This value MUST be a Type found in the Update metadata for this Resource and Class. It is the Update Type.

update-type-version::=*1*2DIGITS . 1*2DIGITS . 1*5DIGITS*

This is the version number of the Update Type metadata. The convention used is a "<major>.<minor>.<release>" numbering scheme. Every time the Update Type metadata changes the version number MUST be increased.

update-type-date:::= *DATE*

The latest change date of the Update Type metadata.

updatetype-field::= <Field Name from Table 11-7>

updatetype-data ::= <valid value as defined in Table 11-7>

An example Update Type section follows:

GetMetadata request:

```
Type: METADATA-UPDATE_TYPE
ID: Property: RES: Add
```

Compact reply:

```
<METADATA-UPDATE_TYPE Resource="Property" Class="RES" Update="Add"
   Version="1.00.000" Date="Sat, 20 Mar 2002 12:03:38 GMT" >
<COLUMNS>→SystemName→Sequence→Attributes→Default→
   ValidationExpressionID→UpdateHelpID→ValidationLookupName→
   ValidationExternalName→</COLUMNS>
<DATA>→STNUM→1→2→→→StNumHelp→→→</DATA>
<DATA>→STNAME→2→2→→→→→StreetName→→→</DATA>
<DATA>→LD→3→2→→ListDate→DateHelp→→→</DATA>
<DATA>→LISTOFF→4→2,3→→→→→→→</DATA>
</METADATA-UPDATE_TYPE>
```

Table 11-7 Metadata Content – Update Type

Metadata Field	Content Type	Description
SystemName	*1*32ALPHANUM*	This is the SystemName of the field as defined in Section 11.3.2.
Sequence	*1*5DIGIT*	Sequence number of the field, representing the order of entry

Table 11-7 Metadata Content – Update Type (continued)

Metadata Field	Content Type	Description
Attributes	1*(**1** | **2** | **3** | **4** | **5** [**,**])	Multiple entries are separated by commas. **Value** / **Meaning** / **Description** 1 / DisplayOnly / Field may not be changed. 2 / Required / Field may not be left blank. 3 / Autopop / Field is populated by the server. 4 / Interactive-Validate / When changed, the client can validate the field only by contacting the server. All fields listed as "AdditionalField" MUST also be passed. 5 / ClearOn-Cloning / The field should be cleared when the containing record is cloned.
Default	*<PLAINTEXT>*	Default value of field (i.e. value if not specified by user)
ValidationExpressionID	*(1*32ALPHANUM ["**,**"])*	<multiple entries are separated by commas> The names of the ValidationExpressions to use. See section 11.4.9
UpdateHelpID	*1*32ALPHANUM*	The name of the entry in the METADATA-UPDATE_HELP table (see Section 11.4.6).
ValidationLookupName	*1*32ALPHANUM*	The name of the ValidationLookup to use. See section 11.4.7
ValidationExternalName	*1*32ALPHANUM*	The name of the ValidationExternal to use. See section 11.4.10

11.4 Metadata Format for Shared Elements

11.4.1 Object

Object type names allow the operator of a particular server to advertise its supported multimedia types. These types are standard MIME types as registered with IANA. RETS does not require that a server make available any particular type of multimedia object. However, a server MUST use a standard well-known name under which to make its multimedia objects available, if a suitable well-known name is defined in the standard. Multimedia names are defined in Table 11-8.

Table 11-8 Well-known Object Types

Object Name	Purpose
Photo	A representation image related to the element defined by the resource KeyField.
Plat	An image of the property boundaries related to the element defined by the resource KeyField
Video	A moving image with or without sound related to the element defined by the resource KeyField.
Audio	A sound clip related to the element defined by the resource KeyField.
Thumbnail	A lower-resolution image related to the element defined by the resource KeyField.

Table 11-8 Well-known Object Types (continued)

Object Name	Purpose
Map	A location image related to the element defined by the resource Key-Field.
VRImage	A multiple-view, possibly-interactive image related to the element defined by the resource KeyField.

The Object metadata starts with a <METADATA-OBJECT> tag with Resource, Version, and Date attributes. This is followed by a <COLUMNS> section, which contains the name of the fields as defined in Table 11-9 followed by the <DATA> section, which contains the actual field information. The Object metadata has the following format:

```
<METADATA-OBJECT · Resource="resource-id" · Version="object-version" ·
    Date="object-date"> ↲
<COLUMNS>→object-field *(→object-field)→</COLUMNS> ↲
*(<DATA>→object-data *(→object-data)→</DATA> ↲)
</METADATA-OBJECT> ↲
```

resource-id ::= *1*32ALPHANUM*

This value MUST be a ResourceID found in the Resource metadata. It is the Resource to which the Objects belong.

object-version ::= *1*2DIGITS . 1*2DIGITS . 1*5DIGITS*

This is the version number of the Object metadata. The convention used is a "<major>.<minor>.<release>" numbering scheme. Every time the Object metadata changes the version number should be increased.

object-date ::= *DATE*

The latest change date of the Object metadata.

object-field ::= <Field Name from Table 11-9>

object-data ::= <valid value as defined in Table 11-9>

An example Resource Object section follows:

GetMetadata request:

```
Class:METADATA-OBJECT
ID:0
```

Compact reply:

```
<METADATA-OBJECT Resource="Property" Version="1.00.000"
    Date="Sat, 20 Mar 2002 12:03:38 GMT" >
<COLUMNS>→ObjectType→StandardName→VisibleName→Description→</COLUMNS>
<DATA>→Photo→image→Full Photos→High Resolution Property Photos→</DATA>
<DATA>→Thumbnail→image→Small Photos→Low  Resolution Property Photos→</
DATA>
</METADATA-OBJECT>
```

Table 11-9 Metadata Content: Resource Object

Metadata Field	Content Type	Description
ObjectType	*1*24ALPHANUM*	The classification of the object. If one of the well-known object types in Table 11-7 applies, then it MUST be used.
MIMEType	*1*24ALPHANUM*	The name of the object type. This is the "mime type" that a client can pass to the "Accept" parameter in the Get Object transaction (see Section 5.1).
VisibleName	*1*32PLAINTEXT*	The user-visible name of the object type.
Description	*1*64PLAINTEXT*	A user-visible description of the object type.

11.4.2 Lookup

This section describes the lookup tables that are referenced by the LookupName in the Table section. There MUST be a corresponding lookup table for every "LookupName".

The Lookup metadata starts with a <METADATA-LOOKUP> tag with Resource, Version, and Date attributes. This is followed by a <COLUMNS> section, which contains the name of the fields as defined in Table 11-10, followed by the <DATA> section, which contains the actual lookup table information. The Lookup metadata has the following format:

```
<METADATA-LOOKUP · Resource="resource-id" · Version="lookup-version" ·
   Date="lookup-date"> ↲
<COLUMNS>→lookup-field *(→lookup-field)→</COLUMNS> ↲
*(<DATA>→lookup-data *(→lookup-data)→</DATA> ↲)
</METADATA-LOOKUP> ↲
```

resource-id ::= *1*32ALPHANUM*

This value MUST be a ResourceID found in the Resource metadata. It is the Resource to which the Lookups belong.

lookup-version ::= *1*2DIGITS . 1*2DIGITS . 1*5DIGITS*

This is the highest version number of any of the contained metadata (Lookup Types). The convention used is a "<major>.<minor>.<release>" numbering scheme. Every time any subordinate document changes the version number MUST be increased.

lookup-date ::= *DATE*

The latest change date of any contained metadata.

lookup-field ::= <Field Name from Table 11-10>

lookup-data ::= <valid value as defined in Table 11-10>

An example Lookup section follows:

GetMetadata request:

```
Type: METADATA-LOOKUP
ID: 0
```

Compact reply:

```
<METADATA-LOOKUP Resource="Property" Version="1.00.000"
   Date="Sat, 20 Mar 2002 12:03:38 GMT" >
<COLUMNS>→LookupName→VisibleName→Version→Date→</COLUMNS>
<DATA>→1→Status→1.00.000→Sat, 20 Mar 2002 12:03:38 GMT→</DATA>
```

```
<DATA>→2→Phone Type→1.00.000→Sat, 20 Mar 2002 12:03:38 GMT→</DATA>
</METADATA-LOOKUP>
<METADATA-LOOKUP Resource="Agent" Version="1.00.000"
    Date="Sat, 20 Mar 2002 12:03:38 GMT">
<COLUMNS>→LookupName→VisibleName→Version→Date→</COLUMNS>
<DATA>→1→Status→1.00.000→Sat, 20 Mar 2002 12:03:38 GMT→</DATA>
</METADATA-LOOKUP>
```

Table 11-10 Metadata Content: Lookup

Field Name	Content Type	Description
LookupName	*1*32ALPANUM*	The name of Lookup Table. There MUST be an entry for each LookupName value used in the Table metadata.
VisibleName	*1*32PLAINTEXT*	A description of the table that is human-readable.
Version	*1*2DIGITS* . *1*2DIGITS* . *1*5DIGITS*	The latest version of this Lookup Table metadata. The convention used is a "<major>.<minor>.<release>" numbering scheme. Clients MAY rely on this data for cache management.
Date	*DATE*	The date on which any of the content of this Lookup was last changed. Clients MAY rely on this date for cache management.

11.4.3 Lookup Type

This section describes the content of a lookup table that is referenced by the LookupName in the Table section. There MUST be a corresponding lookup table for every "Lookup", "LookupMulti", "LookupBitstring" and "LookupBitmask".

The Lookup Type metadata starts with a <METADATA-LOOKUP_TYPE> tag with Resource, Lookup, Version, and Date attributes. This is followed by a <COLUMNS> section, which contains the name of the fields as defined in Table 11-11, followed by the <DATA> section, which contains the actual lookup field information. The Lookup metadata has the following format:

```
<METADATA-LOOKUP_TYPE · Resource="resource-id" ·
    Version="lookup-type-version" ·Date="lookup-type-date"> ↲
<COLUMNS>→lookup-type-field *(→lookup-type-field)→</COLUMNS> ↲
*(<DATA>→lookup-type-data *(→lookup-type-data)→</DATA> ↲)
</METADATA-LOOKUP_TYPE> ↲
```

```
resource-id      ::=   1*32ALPHANUM
```

This value MUST be a ResourceID found in the Resource metadata. It is the Resource to which the Lookup belongs.

```
lookup-name      ::=   1*32ALPHANUM
```

This value MUST be a LookupName found in the Lookup metadata for this Resource. It is the Name of the Lookup table.

```
lookup-type-version::=1*2DIGITS . 1*2DIGITS . 1*5DIGITS
```

This is the version number of the Lookup Type metadata. The convention used is a "<major>.<minor>.<release>" numbering scheme. Every time the Lookup Type metadata changes the version number should be increased.

lookup-type-date::= *DATE*

The latest change date of the Lookup Type metadata.

lookup-type-field::= <Field Name from Table 11-11>

lookup-type-data::= <valid value as defined in Table 11-11>

An example Lookup Type section follows:

GetMetadata request:

```
Type: METADATA-LOOKUP_TYPE
ID: *
```

Compact reply:

```
<METADATA-LOOKUP_TYPE Resource="Property" Lookup="AR" Version="1.00.000"
    Date="Sat, 20 Mar 2002 12:03:38 GMT">
><COLUMNS>→LongValue→ShortValue→Value→</COLUMNS>
<DATA>→Capitol Hill→Cap Hill→1→</DATA>
<DATA>→Juanita  Hill→Juanita→2→</DATA>
<DATA>→Maple Valley→Mpl Valley→3→</DATA>
<DATA>→Downtown Redmond→Dntn Rdmd<4>→</DATA>
</METADATA-LOOKUP_TYPE>
<METADATA-LOOKUP_TYPE Resource="Agent" Lookup="STAT" Version="1.00.000"
    Date= "Sat, 20 Mar 2002 12:03:38 GMT">
<COLUMNS>→LongName→ShortName→Value→</COLUMNS>
<DATA>→Active →ACT→1→</DATA>
<DATA>→Suspended→SUS→2→</DATA>
<DATA>→Inactvie→INA→3→</DATA>
</METADATA-LOOKUP_TYPE>
```

Table 11-11 Metadata Content: Lookup Type

Field Name	Content Type	Description
LongValue	*1*32TEXT*	The value of the field as it is known to the user. This is a localizable, human-readable string. Use of this field is implementation-defined; expected uses include displays on reports and other presentation contexts.
ShortValue	*1*32TEXT*	An abbreviated field value that is also localizable and human-readable. Use of this field is implementation-defined; expected uses include picklist values and other human interface elements.
Value	*1*32ALPHANUM*	The value to be sent to the server when performing a search. This field must be numeric for LookupBitmask and LookupBitstring types. For LookupBitmask fields, $2^{(value-1)}$ is used to compute this component as part of the applicable choices. For LookupBitstring fields, this is the position with in the field, 1-based, at which the value contains a“1”.

11.4.4 Search Help

This section describes the Search Help text tables that are referenced in the Table section. There MUST be a corresponding table entry for each Search HelpTextID referenced in the METADATA-TABLE.

The Search Help metadata starts with a <METADATA-SEARCH_HELP> tag with Resource, Version, and Date attributes. This is followed by a <COLUMNS> section, which

contains the name of the fields as defined in Table 11-12, followed by the <DATA> section, which contains the actual help text. The Search Help Text metadata has the following format:

```
<METADATA-SEARCH_HELP · Resource="resource-id" ·
   Version="search-help-version" ·Date="search-help-date"> ↲
<COLUMNS>→search-help-field *(→search-help-field)→</COLUMNS> ↲
*(<DATA>→search-help-data *(→search-help-data)→</DATA> ↲)
</METADATA-SEARCH_HELP> ↲
```

```
resource-id      ::=   1*32ALPHANUM
```

This value MUST be a ResourceID found in the Resource metadata. It is the Resource to which the Search Help belongs.

```
search-help-version::=1*2DIGITS . 1*2DIGITS . 1*5DIGITS
```

This is the version number of the SearchHelp metadata. The convention used is a "<major>.<minor>.<release>" numbering scheme. Every time the SearchHelp metadata changes the version number MUST be increased.

```
search-help-date::=  DATE
```

The latest change date of the Search Help metadata.

```
search-help-field::= <Field Name from Table 11-12>
```

```
search-help-data::=  <valid value as defined in Table 11-12>
```

An example Search Help definition follows:

GetMetadata request:

```
Type: METADATA-SEARCH_HELP
ID: Property
```

Compact reply:

```
<METADATA-SEARCH_HELP Resource="Property" Version="1.00.000"
   Date="Sat, 20 Mar 2002 12:03:38 GMT" >
<COLUMNS>→SearchHelpID→Value→</COLUMNS>
<DATA>→1→Enter the number in the following format dxd→</DATA>
<DATA>→2→Enter the number in the following format d.dd→</DATA>
</METADATA-SEARCH_HELP>
```

Table 11-12 Metadata Content: Search Help

Field Name	Content Type	Description
SearchHelpID	*1*32ALPHANUM*	A unique ID for the help text. This ID is referenced as the SearchHelpID in section 11.3.2
Value	*1*256TEXT*	The value to be displayed to the user.

11.4.5 Edit Mask

This section describes the Edit Mask table that is referenced in the Table section. There MUST be a corresponding table entry for each Search EditMaskID referenced in the METADATA-TABLE.

A Regular Expression is used to define the edit mask. Table 11-14 describes the structures that make up RETS regulare expressions.

The Editmask metadata starts with a <METADATA-EDITMASK> tag with Resource, Version, and Date attributes. This is followed by a <COLUMNS> section, which contains the name of the fields as defined in Table 11-12, followed by the <DATA> section, which contains the actual help text. The Edit Mask metadata has the following format:

```
<METADATA-EDITMASK · Resource="resource-id" ·
   Version="editmask-version" ·Date="editmask-date"> ↲
<COLUMNS>→editmask-field *(→editmask-field)→</COLUMNS> ↲
*(<DATA>→editmask-data *(→editmask-data)→</DATA> ↲)
</METADATA-EDITMASK> ↲
```

resource-id ::= *1*32ALPHANUM*

This value MUST be a ResourceID found in the Resource metadata. It is the Resource to which the Editmasks belong.

editmask-version::= *1*2DIGITS* **.** *1*2DIGITS* **.** *1*5DIGITS*

This is the version number of the EditMask metadata. The convention used is a "<major>.<minor>.<release>" numbering scheme. Every time the EditMask metadata changes the version number MUST be increased.

editmask-date = *DATE*

The latest change date of the Editmask metadata.

editmask-field ::= <Field Name from Table 11-12>

editmask-data ::= <valid value as defined in Table 11-12>

An example Edit Mask section follows:

GetMetadata request:

```
Type: METADATA-EDITMASK
ID: Property
```

Compact reply:

```
<METADATA-EDITMASK Resource="Property" Version="1.00.000" Date= "Sat, 20 Mar
2002 12:03:38 GMT">
<COLUMNS>→EditMaskID→Value→</COLUMNS>
<DATA>→1→[0-9]{1,2}[x][0-9]{1,2} →</DATA>
<DATA>→2→[0-9]{3}-[0-9]{2}-[0-9}{4} →</DATA>
</METADATA-EDITMASK>
```

Table 11-13 Metadata Content: Edit Mask

Field Name	Content Type	Description
EditMaskID	*1*32ALPHANUM*	A unique ID for the Edit Mask. This ID is referenced as the EditMaskID in section 11.3.2
Value	*1*256TEXT*	The Regular Expression to be used.

RETS Regular Expression Specification

RETS regular expressions are a subset of POSIX 1003.2 extended regular expressions [12], supporting the metacharacters in Table 11-14.

Table 11-14 RETS Regular Expression Metacharacters

Metacharacter	Function
. (period)	Matches any single character
*	Matches zero or more of the preceding pattern
+	Matches one or more of the preceding pattern
?	Matches zero or one of the preceding pattern
|	Alternation: used between two subpatterns, matches either the one to its left or the one to its right.
( ) parentheses	Grouping: causes the enclosed pattern to be treated as atomic. Parentheses may not be nested; that is, only one level of grouping is required.
{*min*[,*max*]} (braces)	Quantifier: matches at least *min* and at most *max* of the preceding pattern, where *min* and *max* are both nonnegative integer values. If *max* is omitted, matches exactly *min* of the preceding pattern.
[ ] brackets	Character class: matches any of the characters contained in the brackets. Except for the circumflex, described below, and the closing bracket, characters within a character class are never treated as metacharacters.
^ (circumflex)	Used as the first character of a character class, reverses the sense of the character class; for example, [^0] matches any character except a "0".
-	Operates only within brackets. Except as the first or last character, denotes a range of characters on the default host collating sequence. For example, [0-9] matches any digit. When - is the first or the last character, it is treated as a member of the character class.
\	Escape: treats the following character as an ordinary character rather than a metacharacter. For example, * matches a single asterisk. The \ character itself must be escaped. The escape character is not needed within character classes.

The following is a simple example:

```
[0-9]+[a-fA-F][1-8][A]?[0-9]{2}[A-C]{1,3}
```

One or more digits, followed by an upper or lower case letter A - F, followed by a digit 1 – 8, optionally followed by one letter A, followed by two digits 0 – 9, followed by between one and three of the letters A – C.

A phone number example:

```
[0-9]{3}-[0-9]{4}
```

11.4.6 Update Help

This section describes the Update Help Text tables that are referenced in the Update Type section of the document. There MUST be a corresponding table entry for each Update Help Text ID referenced in any of the METADATA-UPDATE_TYPEs.

The Update Help metadata starts with a <METADATA-UPDATE_HELP> tag with Resource, Version, and Date attributes. This is followed by a <COLUMNS> section, which

contains the name of the fields as defined in Table 11-15, followed by the <DATA> section, which contains the actual help text. The Update Help Text metadata has the following format:

```
<METADATA-UPDATE_HELP · Resource="resource-id" ·
   Version="update-help-version" ·Date="update-help-date"> ↲
<COLUMNS>→update-help-field *(→update-help-field)→</COLUMNS> ↲
*(<DATA>→update-help-data *(→update-help-data)→</DATA> ↲)
</METADATA-UPDATE_HELP> ↲
```

resource-id ::= *1*32ALPHANUM*

This value MUST be a ResourceID found in the Resource metadata. It is the Resource to which the Update Help belongs.

update-help-version::=*1*2DIGITS . 1*2DIGITS . 1*5DIGITS*

This is the version number of the Update Help metadata. The convention used is a "<major>.<minor>.<release>" numbering scheme. Every time the Update Help metadata changes the version number MUST be increased.

update-help-date::= *DATE*

The latest change date of the Update Help metadata.

update-help-field::= <Field Name from Table 11-15>

update-help-data::= <valid value as defined in Table 11-15>

An example Update Help section follows:

GetMetadata request:

```
Type: UPDATE_HELP
ID: Property
```

Compact reply:

```
<METADATA-UPDATE_HELP Resource="Property" Version="1.00.000"
   Date="Sat, 20 Mar 2002 12:03:38 GMT" >
<COLUMNS>→UpdateHelpID→Value→</COLUMNS>
<DATA>→1→Enter the number in the following format dxd→</DATA>
<DATA>→2→Enter the number in the following format d.dd→</DATA>
</METADATA-UPDATE_HELP>
```

Table 11-15 Metadata Content: Update Help

Field Name	Content Type	Description
UpdateHelpID	*1*32ALPHANUM*	A unique ID for the help text. This ID is referenced as the UpdateHelpID in section 11.4.6.
Value	*1*256TEXT*	The value to be displayed to the user.

11.4.7 Validation Lookup

This section describes the Validation Lookup tables that are referenced in the Update Type section of the document. There MUST be a corresponding Validation Lookup Table for each one referenced in the METADATA-UPDATE_TYPEs.

The Validation Lookup metadata starts with a <METADATA-VALIDATION_LOOKUP> tag with Resource, Version, and Date attributes. This is followed by a <COLUMNS> section, which contains the name of the fields as defined in Table 11-16, followed by the

<DATA> section, which contains the actual field information. The Validation Lookup metadata has the following format:

```
<METADATA-VALIDATION_LOOKUP · Resource="resource-id" ·
    Version="valid-lookup-version" ·Date="valid-lookup-date"> ↲
<COLUMNS>→valid-lookup-field *(→valid-lookup-field)→</COLUMNS> ↲
*(<DATA>→valid-lookup-data *(→valid-lookup-data)→</DATA> ↲)
</METADATA-VALIDATION_LOOKUP> ↲
```

resource-id ::= *1*32ALPHANUM*

This value MUST be a ResourceID found in the Resource metadata. It is the Resource to which the Validation Lookups belong.

valid-lookup-version=*1*2DIGITS* . *1*2DIGITS* . *1*5DIGITS*

This is the highest version number of any of the contained metadata (Lookup Types). The convention used is a "<major>.<minor>.<release>" numbering scheme. Every time any contained metadata changes the version number MUST be increased.

valid-lookup-date::= *DATE*

The latest change date of the contained metadata.

valid-lookup-field::=<Field Name from Table 11-16>

valid-lookup-data::= <valid value as defined in Table 11-16>

An example Validation Lookup section follows:

GetMetadata request:

```
Type: METADATA-VALIDATION_LOOKUP
ID: Property
```

Compact reply:

```
<METADATA-VALIDATION_LOOKUP Resource="Property" Version="1.00.000"
    Date= "Sat, 20 Mar 2002 12:03:38 GMT" >
<COLUMNS>→ValidationLookupName→Parent1Field→ Parent2Field→
    Version→Date→</COLUMNS>
<DATA>→School→Area→Subarea→1.00.000→Sat, 20 Mar 2002 12:03:38 GMT →</
DATA>
<DATA>→ZipCode→Area→→1.00.000→Sat, 20 Mar 2002 12:03:38 GMT →</DATA>
<DATA>→City→→→1.00.000→Sat, 20 Mar 2002 12:03:38 GMT →</DATA>
</METADATA-VALIDATION_LOOKUP>
```

Table 11-16 Metadata Content: Validation Lookup

Field Name	Content Type	Description
ValidationLookup-Name	*1*32ALPHANUM*	The unique name of this Validation Lookup. Each Name in the Update Type ValidationLookupName field MUST have a definition.
Parent1Field	*1*32ALPHANUM*	If a value is present, it is a SystemName field in the same table as defined in Section 11.3.2 and indicates a dependency on this field.
Parent2Field	*1*32ALPHANUM*	If a value is present it is a SystemName field in the same table as defined in Section 11.3.2 and indicates an additional dependency on this field.

Table 11-16 Metadata Content: Validation Lookup (continued)

Field Name	Content Type	Description
Version	*1*2DIGITS* . *1*2DIGITS* . *1*5DIGITS*	The version of this Validation Lookup metadata. The convention used is a "<major>.<minor>.<release>" numbering scheme. Clients MAY rely on this data for cache management.
Date	*DATE*	The date on which any of the content of this Validation Lookup metadata was last changed. Clients MAY rely on this date for cache management.

11.4.8 Validation Lookup Type

This section describes the content of the Validation Lookup tables that are referenced in the Table section of the document. There MUST be a corresponding Validation Lookup Type table for each one referenced in the METADATA-UPDATE_TYPE.

The Validation Lookup Type provides a list of all the valid values for a field. This is different than the Lookup described in Section 11.4.2. The Validation Lookup is used for two cases: 1) the list is too long to be provided as a standard lookup (e.g. Street Name) and 2) there is a dependency on the value in another field. For example, a valid entry for a School District might depend on the Area and SubArea that is entered.

The Validation Lookup Type metadata starts with a <METADATA-VALIDATION_LOOKUP_TYPE> tag with Resource, ValidationLookup, Version, and Date attributes. This is followed by a <COLUMNS> section, which contains the name of the fields as defined in Table 11-17, followed by the <DATA> section, which contains the actual valid field information. If the parent fields are undefined then any value in the Validation Lookup is acceptable. The Validation Lookup metadata has the following format:

```
<METADATA-VALIDATION_LOOKUP_TYPE · Resource="resource-id" ·
    Version="valid-lookup-type-version" ·
    Date="valid-lookup-type-date"> ↲
<COLUMNS>→valid-lookup-type-field *(→valid-lookup-type-field)→
    </COLUMNS> ↲
*(<DATA>→valid-lookup-type-data *(→valid-lookup-type-data)→
    </DATA> ↲)
</METADATA-VALIDATION_LOOKUP_TYPE> ↲
```

resource-id ::= 1*32ALPHANUM

This value MUST be a ResourceID found in the Resource metadata. It is the Resource to which the Validation Lookup Type belongs.

valid-lookup-type-name::=*1*32ALPHANUM*

This value MUST be a ValidationLookupName found in the Validation Lookup metadata for the Resource. It is the name of the Validation Lookup Type.

*valid-lookup-type-version::=1*2DIGITS . 1*2DIGITS . 1*5DIGITS*

This is the version number of this Validation Lookup Type metadata. The convention used is a "<major>.<minor>.<release>" numbering scheme. Every time this Validation Lookup Type metadata changes the version number should be increased.

valid-lookup-type-date::=*DATE*

The latest change date of the Validation Lookup Type metadata.

valid-lookup-type-field::=<Field Name from Table 11-17>

valid-lookup-type-data::=<valid value as defined in Table 11-17>

An example Validation Lookup Type section follows:

GetMetadata request:

```
Type: METADATA-VALIDATION_LOOKUP_TYPE
ID: Property: School
```

Compact reply:

```
<METADATA-VALIDATION_LOOKUP_TYPE Resource="Property"
    ValidationLookup="School" Version="1.00.000"
    Date="Sat, 20 Mar 2002 12:03:38 GMT" >
<COLUMNS>→ValidText→Parent1Value→ Parent2Value→</COLUMNS>
<DATA>→133→AREA1→SUBAREA1→</DATA>
<DATA>→134→AREA1→SUBAREA2→</DATA>
<DATA>→135→AREA2→→</DATA>
</METADATA-VALIDATION_LOOKUP_TYPE>
```

Table 11-17 Metadata Conent: Validation Lookup Type

Field Name	Content Type	Description
ValidText	*1*32ALPHANUM*	A valid value for the field.
Parent1Value	*1*32ALPHANUM*	If this field is present then the ValidText can be used if the data in the Parent1 field is set to this value. If Parent1 is present in the PARENTFIELDS tag then this field is required.
Parent2Value	*1*32ALPHANUM*	If this field is present then the ValidText can be used if the data in the Parent2 field is set to this value. If Parent2 is present in the PARENTFIELDS tag then this field is required.

11.4.9 Validation Expression

This section describes the ValidationExpression table that is referenced in Section 11.3.4. There MUST be a corresponding table entry for each ValidationExpressionID referenced in the METADATA-UPDATE_TYPEs for a Resource.

The table contains expressions that are to be evaluated when a field value is entered by the user. Expressions in the list MUST be evaluated in the order in which they appear in the list. There are three types of validation expressions, each introduced by a reserved token preceding the expression, given in Table 11-18:

Table 11-18 Validation Expression Types

Keyword	Type	Purpose
ACCEPT	Boolean	If the expression is true, the field value is considered accepted without further testing. Subsequent SET expressions MUST be executed.
REJECT	Boolean	If the expression is true, the field value is considered rejected without further testing. Subsequent SET expressions MUST NOT be evaluated.
SET	Assignment	The expression MUST begin with a field name and an equal sign (“=”). The following expression is evaluated and the result stored in the designated field.

Expressions are algebraic formulas containing keywords and operators. Expressions may contain parentheses, and consist of keywords representing any of:

- The current value of any field in the input list
- The current value of any Well-Known Name field in the user's agent record that is returned in the response to the login transaction (see 4.9, "Well-Known Names").
- Literal values.
- A special token (Table 11-18 Metadata Content – Validation Expression Special Operand Tokens).

together with the operators in Table 11-19. Arithmetic operations MUST be carried out using IEEE-754 arithmetic with a representation of at least 64 bits. Comparison operations on strings MUST use simple binary collation. If an error or arithmetic exception occurs during expression evaluation, field value is considered erroneous, regardless of the expression type.

Table 11-19 Validation Expression Operators

Operator	Precedence	Operation
/, *, **.MOD.**	1	Division, multiplication, and remainder (modulo)
+,−	2	Addition and subtraction, applied as follows: 1. If both operands are numeric, the operation is algebraic. 2. If either operand is a string, it is converted to numeric and the operation is algebraic. If an error occurs during the conversion, the field value MUST be rejected. 3. For "+", if either operand is a date, the other must be an integer, a string that can be converted to an integer, or a string representing an interval in ISO8601 format. If no conversion is possible, the field value MUST be rejected 4. For "-", if the left operand is a date or time, the other operand must be a date, a time, or a string representing an interval, and the result must be a string representing an interval in ISO8601 format.
.CONTAINS.	2	A Boolean operator taking strings as its left and right operands. The operation is TRUE if the left operand contains the right operand as a substring anywhere within it.
<, >, <=, >=,	3	Comparison operators with their conventional meaning. If one operand is numeric and the other is a string, the string MUST be converted to a number prior to the comparison. If an error occurs during the conversion, the field value must be rejected.
=, **!=**	4	Comparison operators with their conventional meaning. If one operand is numeric and the other is a string, the string MUST be converted to a number prior to the comparison. If an error occurs during the conversion, the field value must be rejected.
.AND.	5	A Boolean operator that takes two Boolean operands, and whose value is TRUE if and only if both of its operands are TRUE.
.OR.	6	A Boolean operator that takes two Boolean operands, and whose value is TRUE if either of its operands is TRUE.
.NOT.	7	A Boolean operator that takes a single Boolean operand and returns its inverse.

Literal values to be compared against dates or times are expressed in the ISO8601 format.

Table 11-20 Validation Expression Special Operand Tokens

Token	Value
.TODAY.	The current date.
.NOW.	The current time.
.ENTRY.	The current field text, as a string.
.EMPTY.	A value that matches an empty or all-blank field. Supplies an empty (zero-length) field when used in a SET expression.
.OLDVALUE.	The text that was in the field as returned from the host in the search operation. If the field is new, .OLDVALUE. is an empty string.
.USERID.	The value of the user-id field returned in the Login transaction (Section 4.9).
.USERCLASS.	The value of the user-class field returned in the Login transaction (Section 4.9).
.USERLEVEL.	The value of the user-level field returned in the Login transaction (Section 4.9).
.AGENTCODE.	The value of the agent-code field returned in the Login transaction (Section 4.9).
.BROKERCODE.	The value of the broker-code field returned in the Login transaction (Section 4.9).
.BROKERBRANCH.	The value of the broker-branch field returned in the Login transaction (Section 4.9).

The Validation Expression metadata starts with a <METADATA-VALIDATION_EXPRESSION> tag with Resource, Version and Date attributes. This is followed by a <COLUMNS> section, which contains the name of the fields as defined in Table 11-21, followed by the <DATA> section, which contains the actual Validation Expressions. The Validation Expression metadata has the following format:

```
<METADATA-VALIDATION_EXPRESSION · Resource="resource-id" ·
    Version="valid-expression-type-version" ·
    Date="valid-expression-type-date"> ↲
<COLUMNS>→valid-expression-type-field
    *(→valid-expression-type-field)→</COLUMNS> ↲
*(<DATA>→valid-expression-type-data *(→valid-expression-type-data)→
    </DATA> ↲)
</METADATA-VALIDATION_EXPRESSION> ↲

resource-id      ::=   1*32ALPHANUM
```

This value MUST be a ResourceID found in the Resource metadata. It is the Resource to which the Objects belong.

```
valid-expression-version::=1*2DIGITS . 1*2DIGITS . 1*5DIGITS
```

This is the version number of the Validation Expression metadata. The convention used is a "<major>.<minor>.<release>" numbering scheme. Every time the Validation Expression metadata changes the version number should be increased.

```
valid-expression-date:=DATE
```

The latest change date of the ValidationExpression metadata.

```
valid-expression-field::=<Field Name from Table 11-21>
```

valid-expression-data::=<expression type keyword><valid validation expression>

Table 11-21 Metadata Content: Validation Expression

Field Name	Content Type	Description
Validation-ExpressionID	*1*32ALPHANUM*	A unique ID for the ValidationExpression. This ID is referenced as the ValidationExpression in Section 11.3.4.
Validation-ExpressionType	*1*32ALPHANUM*	A validation expression type from Table 11-18.
Value	*1*512TEXT*	The test expression to be evaluated.

An example Validation Expression section follows:

GetMetadata request:

```
Type: METADATA-VALIDATION_EXPRESSION
ID: Property
```

Compact reply:

```
<METADATA-VALIDATION_EXPRESSION Resource="Property" Version="1.00.000"
    Date= "Sat, 20 Mar 2002 12:03:38 GMT" >
<COLUMNS>→ValidationExpressionID→ValidationExpressionType→Value→</COLUMNS>
<DATA>→Office1→ACCEPT>→
     LAG=.AGENTCODE. .OR. (LO=.BROKERCODE. .AND. .ENTRY.=OFFICE)→</DATA>
<DATA>→Agent1→ACCEPT→(LAG=.AGENTCODE). .OR. (SAG=.AGENTCODE.)→</DATA>
<DATA>→ListDate→ACCEPT→ LD>.TODAY. - 3 .AND. LD<.TODAY. + 3→</DATA>
</METADATA-VALIDATION_EXPRESSION>
```

11.4.10 Validation External

This section describes the Validation External tables that are referenced in the Update Type section of the document. There MUST be a corresponding Validation External table for each one referenced in any of the METADATA-UPDATE_TYPEs for the Resource.

The Validation External metadata starts with a <METADATA-VALIDATION_EXTERNAL> tag with Resource, Version, and Date attributes. This is followed by a <COLUMNS> section, which contains the name of the fields as defined in Table 11-22, followed by the <DATA> section, which contains the actual field information. The Validation External metadata has the following format:

```
<METADATA-VALIDATION_EXTERNAL · Resource="resource-id" ·
    Version="validation-external-version" ·
    Date="validation-external-date"> ↲
<COLUMNS>→validation-external-field *(→validation-external-field)→
    </COLUMNS> ↲
*(<DATA>→validation-external-data *(→validation-external-data)→
    </DATA> ↲)
</METADATA-VALIDATION_EXTERNAL> ↲
```

resource-id ::= *1*32ALPHANUM*

This value MUST be a ResourceID found in the Resource metadata. It is the Resource to which the Validation Externals belong.

validation-external-version::=*1*2DIGITS . 1*2DIGITS . 1*5DIGITS*

This is the highest version number of any of the contained metatadata (Validation External Types). The convention used is a "<major>.<minor>.<release>" numbering

scheme. Every time any contained metadata changes the version number MUST be increased.

validation-external-date::=*DATE*

The latest change date of the contained metadata.

validation-external-field::=<Field Name from Table 11-22>

validation-external-data::=<valid value as defined in Table 11-22>

An example Validation External section follows:

GetMetadata request:

```
Type: METADATA-VALIDATION_EXTERNAL
ID: Property
```

Compact reply:

```
<METADATA-VALIDATION_EXTERNAL Resource="Property" Version="1.00.000"
    Date= "Sat, 20 Mar 2002 12:03:38 GMT" >
<COLUMNS>→ValidationExternalName→SearchResource→SearchClass→Version→Date→
</COLUMNS>
<DATA>→1→Office→ Office→1.00.000→Sat, 20 Mar 2002 12:03:38 GMT →</DATA>
<DATA>→2→Tax→HENN→1.00.000→Sat, 20 Mar 2002 12:03:38 GMT →</DATA>
</METADATA-VALIDATION_EXTERNAL>
```

Table 11-22 Metadata Content: Validation External

Field Name	Content Type	Description
ValidationExternal-Name	*1*32ALPHANUM*	The unique name of this Validation External. Each Name in the Update Type ValidationExternalName field MUST have a definition.
SearchResource	*1*32ALPHANUM*	The ResourceID of the Resource to be searched from 11.2.2.
SearchClass	*1*32ALPHANUM*	The ClassName within the Resource to be searched from 11.3.1.
Version	*1*2DIGITS "." 1*2DIGITS "." 1*5DIGITS*	The latest version of this Validation External metadata. The convention used is a "<major>.<minor>.<release>" numbering scheme. Clients MAY rely on this data for cache management.
Date	*DATE*	The date on which any of the content of this Validation External was last changed. Clients MAY rely on this date for cache management.

11.4.11 Validation External Type

This section describes the content of the Validation External Type tables that are referenced in the Table section of the document. There MUST be a corresponding Validation External Type table for each one referenced in the METADATA-UPDATE_TYPEs for the Resource.

The Validation External Type provides lists of search, display, and results fields. The Validation External may be used for several cases: 1) The database involved is too large or dynamic to be provided as a standard lookup (e.g. Tax). 2) There are business rules that can only be enforced on the server (e.g. expiration dates). 3) The content of a field

populates fields from another database (e.g. Sale_agent_name, Sale_office_name, Sale_office_id from Sale_agent_id).

The metadata starts with a <METADATA-VALIDATION_EXTERNAL_TYPE> tag with Resoure, ValidationExternal, Version, and Date attributes. This is followed by a <COLUMNS> section, which contains the name of the fields as defined in Table 11-23, followed by the <DATA> section, which contains the actual valid field information. The Validation External Type metadata has the following format:

```
<METADATA-VALIDATION_EXTERNAL_TYPE · Resource="resource-id" ·
    Version="valid-ext-type-version" · Date="valid-ext-type-date"> ↲
<COLUMNS>→valid-ext-type-field *(→valid-ext-type-field)→</COLUMNS>
↲
*(<DATA>→valid-ext-type-data *(→valid-ext-type-data)→</DATA> ↲)
</METADATA-VALIDATION_EXTERNAL_TYPE> ↲
```

resource-id ::=*1*32ALPHANUM*

This value MUST be a ResourceID found in the Resource metadata. It is the Resource to which the Validation External Type belongs.

valid-ext-type-name::=*1*32ALPHANUM*

This value MUST be a ValidationExternalName found in the Validation External metadata for the Resource. It is the name of the Validation External Type.

valid-ext-type-version::=*1*2DIGITS . 1*2DIGITS . 1*5DIGITS*

This is the version number of the Validation External Type metadata. The convention used is a "<major>.<minor>.<release>" numbering scheme. Every time the Validation External Type metadata changes the version number should be increased.

valid-ext-type-date::=*DATE*

The latest change date of the Validation External Type metadata.

valid-ext-type-field::=<Field Name from Table 11-23>

valid-ext-type-data::=<valid value as defined in Table 11-23>

An example Validation External Type section follows:

GetMetadata request:

```
Type: METADATA-VALIDATION_EXTERNAL_TYPE
ID: Property: VET1
```

Compact reply:

```
<METADATA-VALIDATION_EXTERNAL_TYPE Resource="Property"
    ValidationExternal="VET1" Version="1.00.000"
    Date="Sat, 20 Mar 2002 12:03:38 GMT" >
<COLUMNS>→SearchField→DisplayField→ResultsFields→</COLUMNS>
<DATA>→AgentID, AgentCode→AgentName, OfficeName→SaleAgentID=AgentID,
    SaleAgentName=AgentName, SaleOfficeID=OfficeID,
    SaleOfficeName=OfficeName→</DATA>
</METADATA-VALIDATION_EXTERNAL_TYPE>
```

Table 11-23 Metadata Content: Validation External Type

Field Name	Content Type	Description
SearchField	*1*512PLAINTEXT*	A comma separated list of valid fields using SystemName from Section 11.3.2.
DisplayField	*1*512PLAINTEXT*	A comma separated list of valid fields using SystemName from Section 11.3.2.
ResultFields	*1*1024PLAINTEXT*	A comma separated list of valid field pairs joined by = (equal) the first is a target field in the table being updated and the second is a source field in the table being searched. The fields use a SystemName from Section 11.3.2.

SECTION

12

GETMETADATA TRANSACTION

The GetMetadata transaction is used to retrieve structured information known as metadata related to the system entities. Metadata requested and returned from this transaction are requested and returned as MIME media types.

12.1 Required Client Request Header Fields

In addition to the Required Client Request Header Fields specified in Section 3.4, the header of any messages sent from the client MUST contain the following header fields:

Accept — The client MUST request a media <type> using the standard HTTP Accept header field. Media-type formats (subtypes) are registered with the Internet Assigned Number Authority (IANA) and use a format outlined in RFC 2045 [8]. When submitting a request the client MUST specify the desired type and format. If the server is unable to provide the desired format it SHOULD return a "406 Not Acceptable" status. However, if there are no objects of any <subtype> available for the requested object the server SHOULD return "404 Not Found". The format of the Accept field is as follows:

Accept	::=	**Accept :** *type* **/** *subtype* [**;** *parameter*] *(**,** *SP type* **/** *subtype* [**;** *parameter*])
type	::=	***** | **text**
subtype	::=	***** | <A publicly-defined extension token that has been registered with IANA>
parameter	::=	**q =** < qvalue scale from 0 to 1 >

A compliant server MUST support at least text/plain and text/xml.

A more complete list is available at:

ftp.isi.edu/in-notes/iana/assignments/media-types

The `qvalue` is used to specify the desirability of a given media type/format, with "1" being the most desirable, "0" being the least desirable, and a range in between. The default `qvalue` is "1".

Example: `Accept: text/xml,text/plain;q=0.5`

Verbally, this would be interpreted as "text/xml is the preferred media type, but if that does not exist, then send the text/plain entity."

12.2 Required Request Arguments

`Type` ::= <A grouping of related metadata elements (see Section 11)>

The type of metadata being requested. The Type MUST begin with METADATA and MAY be one of the defined metadata types (see Section 11).

`ID` ::= *metadata-id*[`:` metadata-id]

metadata-id ::= *1*ALPHANUM* | *

Metadata is organized hierarchically. Each level specifies in its first field an identifier for the metadata contained within that level (e.g. for the Resource level: ResourceID--Agent, Property, etc. for the Lookup level: LookupName—Status, Area, etc.). This identifier can be used to restrict requests to the Type metadata contained within specific instances of higher levels. If the last metadata-id is 0 (zero), then the request is for all Type metadata contained within that level; if the last metadata-id is "*", then the request is for all Type metadata contained within that level and all metadata Types contained within the requested Type.

Note: The metadata-id for METADATA-SYSTEM and METADATA-RESOURCE must be 0 or *.

12.3 Optional Request Arguments

Format = **COMPACT** | **STANDARD-XML** | **STANDARD-XML:***version*

version ::= <RETS metadata DTD version>

"COMPACT" means a table descriptor , field list <COLUMNS> followed by a delimited set of the data fields. See Section 11 for more information on the COMPACT formats. "STANDARD-XML" means an XML presentation of the data in the format defined by the RETS Metadata XML DTD. Servers MUST support all formats. If the format is not specified, the STANDARD-XML presentation will be returned.

When the client requests the STANDARD-XML representation, it may also specify a DTD version. The server SHOULD be prepared to support at least the current version and the prior version. Metadata DTD versions are of the form

`RETS-METADATA-`*yyyymmdd*`.dtd`

where *yyyymmdd* is the release date of the DTD.

12.4 Required Server Response Header Fields

In addition to the other Required Server Header Fields specified in Section 3.3 the following response header fields are required.

Content-Type	The media type of the underlying data. The server MUST return this field in all replies. This field MUST be set to the type of media returned. See Section 12.1 for more information on <type> and <subtype>.

Content-Type ::= **Content-Type :** *type / subtype*

Example: `Content-Type: text/plain`

Content ID — An ID for the object. This field MUST be returned with the value of the first METADATA-component returned.

Content-ID ::= **Content-ID :** **64*<TEXT, excluding CR/LF>

Example: `Content-ID: METADATA-SYSTEM`

MIME-Version — All responses MUST include a MIME-Version of "1.0" in the response header.

Example: `MIME-Version: 1.0`

12.5 Optional Server Response Header Fields

In addition to the other Optional Server Header Fields specified in Section 3.8 the following response header field are also optional.

Description — A text description of the object.

Description ::= **Content-Description :** **64*<TEXT, excluding CR/LF>

Example: `Content-Description: NTREIS`

12.6 Required Response Arguments

There are no required response arguments.

12.7 Optional Response Arguments

There are no optional response arguments.

12.8 Metadata Response Body Format

The body of the metadata response has the following format when replying to a request with the format set to"COMPACT":

<RETS *1*SP* **ReplyCode=***quoted-reply-code 1*SP*
ReplyText=*quoted-string *SP* **>** *CRLF*
[**metadata-segment*]
[*rets-status-tag*]
</RETS> *CRLF*

metadata-segment::= <A metadata segment as defined in Section 11.>

The body of the metadata response has the following format when replying to a format request of "STANDARD-XML" data:

<?xml version="1.0" ?>
[*doctype***]**
<RETS *1*SP* **ReplyCode=***quoted-reply-code 1*SP*
ReplyText=quoted-string *SP >
[**XML-metadata-segment***]**
[*rets-status-tag* **]**
</RETS> *CRLF*

doctype ::= **<!DOCTYPE RETS SYSTEM "***dtd-version***">**

dtd-version ::= <Name of the RETS Metadata DTD used to produce this document>

XML-metadata-segment::=A metadata segment as defined by the RETS Metadata XML DTD.

12.9 Metadata

A full description of the Metadata Dictionary is provided in Section 11.

12.10 Reply Codes

Table 12-1 GetMetadata Reply Codes

Reply Code	Meaning
20500	Invalid Resource The request could not be understood due to an unknown resource.
20501	Invalid Type The request could not be understood due to an unknown metadata type.
20502	Invalid Identifier The identifier is not known inside the specified resource.
20503	No Metadata Found No matching metadata of the type requested was found.
20506	Unsupported Mimetype The server cannot return the metadata in any of the requested MIME types.
20507	Unauthorized Retrieval The metadata could not be retrieved because it requests metadata to which the supplied login does not grant access (e.g. Update Type data).
20508	Resource Unavailable The requested resource is currently unavailable.
20509	Metadata Unavailable The requested metadata is currently unavailable.
20510	Request Too Large Metadata could not be retrieved because a system limit was exceeded.
20511	Timeout The request timed out while executing.
20512	Too many outstanding requests The user has too many outstanding requests and new requests will not be accepted at this time.
20513	Miscellaneous error The server encountered an internal error.
20514	Requested DTD version unavailable. The client has requested the metadata in STANDARD-XML format using a DTD version that the server cannot provide.

SECTION

13

COMPACT DATA FORMAT

Clients may choose to access data from a server in a "COMPACT" data format that does not use full XML representation. When a client requests information from a compliant server in "COMPACT" format, it will typically need to interpret the result by using the metadata that the server makes available.

13.1 Overall format

Compact-format records are sequences of fields separated by delimiter. A tab character (an octet with a binary value of 9) is the default delimiter unless another is specified as part of the transaction. The sequence of fields MUST be described by a <COLUMNS> tag in the body of the message that carries the compressed records. No field may be omitted from the <DATA>; if the value of a particular field for some record is undefined, the value SHOULD be represented by two delimiters with no intervening space.

Compact records are enclosed within a <DATA> start tag and a </DATA> end tag. The records are separated from each other by a CRLF line termination sequence.

13.2 Decoded Format

Compact-decoded format requires sending data in its most people-readable form. Coded data is data that is stored as an enumeration, multivalue, boolean, abbreviation, or arbitrary string with its meaning defined elsewhere in the system. At minimum, a server SHOULD perform the lookup or expansion from the lookup or validation-lookup values defined in the metadata for that field but it MAY be a richer value provided by the system's reporting capabilities. If the field is multivalued, commas and a space separate the decoded values.

13.3 Transmission standards

A client or server transmitting a compact record MUST encode the data according to Table 13-1.

Table 13-1 Compact Data Format Representation

Type	Encoding Format
Numeric	An optional sign, followed by zero or more digits, followed by an optional period, followed optionally by zero or more digits. A valid number MUST contain at least one digit if it includes a decimal point or sign. The value may contain leading zeros before the period and/or trailing zeros after the decimal point and fraction, if any. Data types Tiny, Small, Int and Long (Table 11-5) may be signed but may not have nonzero digits after the decimal point. Values with the interpretation LookupBitmask must not be signed, nor may they have nonzero digits after the decimal point.
Character	The plain character sequence, except for LookupMulti, which contains multiple sequences of characters separated by commas. Values with the interpretation LookupBitstring must conain only the characters “0” and “1”.
Date	Eight digits in YYYY-MM-DD order, with dashes separating the year from the month and the month from the day.
Time	Six digits in hh:mm:ss[.sss], with colons separating the hour from the minute and the minute from the second, with a three-digit optional fractions of a second format separated from the seconds with a decimal <".">.
Date-Time	A fourteen-digit string with separators as above, and a space between the day and the hour, as `YYYY-MM-DDThh:mm:ss[.sss]`, with a three-digit optional fractions of a second separated from the seconds with a decimal <".">
MultiSelect	A string consisting of one or more substrings, comma-delimited, each of which corresponds to an entry in the field’s associated MetadataLookup table.
Boolean	A single character, either 1 for true or 0 for false.

SECTION

14

SESSION PROTOCOL

A RETS session follows a well-defined timing sequence in becoming established and in terminating. In particular, the authorization sequence MUST be followed in order to begin using other transactions within the protocol. The protocol contains four phases: connection establishment, authorization, session and termination.

14.1 Connection Establishment

A client initiates communication with a server by beginning a TCP connection on any mutually agreed TCP port, with the default being 6103 for unencrypted connections, and port 443 for SSL-encrypted connections. When the TCP connection has entered the Established state, the session proceeds to the start of the Authorization phase.

14.2 Authorization

Authorization begins when the client sends the server a Login transaction. The Login transaction contains the basic information that the server requires in order to start an authorization decision: the user ID and optionally, some information about the client software.

A server responds to the Login request by sending back a "401 Unauthorized" status code and a WWW-Authenticate header. This is part of an authentication challenge to the client. Part of the WWW-Authenticate header may contain a checksum (nonce) of a concatenation of the following:

1. The client-IP.
2. The server-supplied timestamp.
3. The server's private-key.

Server implementers should note that because of intervening proxy servers, the client IP address may change from connection to connection.

The client concatenates the nonce to the checksum of the Request-URI; then performs an MD5 digest using a concatenation of the username, realm and password as the secret. This result is then returned to the server as part of an Authorization header. The server MUST then compute the equivalent function using its own stored copy of the user's password. If the two match and the nonce is the same, the user is considered authenticated, and the

login can proceed with the server informing the client of the available capabilities. The login has been accomplished without actually sending the password. A server MAY provide an anonymous login. A client wishing an anonymous login sends an empty Authentication field in its Login transaction, after which the authorization proceeds as before.

14.3 Session

Once the Authorization phase has been completed, both endpoints enter the Session phase. During the Session phase, clients may issue any combination of requests for which they are authorized. The first of these MUST be to issue a GET requests for the "Action" URL, if any, included in the Login response (Section 4.10). After this, clients may issue other transactions.

Clients MAY issue multiple transactions without waiting for responses. However, servers are not required to process these requests in parallel, nor are servers required to complete the requests in the order in which they were issued. If a client issues a request before receiving a response to some earlier request, the client MUST be prepared to receive the responses in any order. The only way for a client to guarantee sequential execution of requests on every server is to wait for a response to any outstanding request before issuing a new request.

14.4 Termination

A client SHOULD initiate termination of the session by sending a Logoff transaction. If a server receives a Logoff transaction while other operations are pending, it SHOULD abort those pending operations. However, a server MUST NOT rely on receiving a Logoff transaction in order to terminate a session, due to the possibility of communications problems preventing the transmission of the Logoff transaction by the client.

Servers SHOULD provide a timeout mechanism, and if they do, MUST inform the client of the timeout interval during the Login transaction (Section 4.7).

SECTION

15

SAMPLE SESSIONS

To be supplied.

SECTION

16

ACKNOWLEDGMENTS

The creation of this specification would not have been possible without the sponsorship and coordination of efforts provided by the National Association of REALTORS®.

This document has benefited greatly from the comments of all those participating in the National Association of REALTORS®-Standards Work Group.

In addition to the authors, valuable discussion instrumental in creating this document has come from:

Richard Mendenhall
National Association of REALTORS®

Dale Stinton
National Association of REALTORS®

Larry Colson
Moore Data Management Services

Tom Curtis
Metro MLS

Kevin Knoepp
GTE Enterprise Solutions

Tom McLean
Resolution Software Consulting, Inc.

Tony Salvati
Grant Thornton

Errol Samuelson
RealSelect, Inc.

Allan Shapiro
Interealty Corporation

Stuart Schuessler
MarketLinx Corporation

Michael DelGaudio
MRIS, Inc.

Mark Lesswing
National Association of Realtors®

SECTION

17

AUTHORS

Leo Bijnagte
Vista Information Systems
100 Washington Square, Suite 1000
Minneapolis, MN 55401

Email: leob@fnis.com

Dan Musso
WyldFyre Technologies, Inc.
900 East Hamilton Ave.
Suite 500
Campbell, CA 95008

Email: dan@WyldFyre.com

Bruce Toback
OPT, Inc.
11801 N. Tatum Blvd.
Suite 142
Phoenix, AZ 85028

Email: btoback@optc.com

SECTION

18

REFERENCES

[1] Braden, R., "Requirements for Internet Hosts — Communication Layers" STD 3, RFC 1123, IETF 1989.

[2] Fielding, R., "Hypertext Transfer Protocol — Version 1.1", RFC 2616, January 1997

[3] Rivest, R., "The MD5 Message Authentication Algorithm", RFC 1321, April 1992

[4] Crocker, D., "Standard for ARPA Internet Text Messages", RFC 822, IETF 1982

[5] US-ASCII. Coded Character Set - 7-Bit American Standard Code for Information Interchange. Standard ANSI X3.4-1986, ANSI, 1986.

[6] Franks, J., Hallam-Baker, P., Hostetler, J., Leach, P., Luotonen, A., Sink, E., and L. Stewart, "An Extension to HTTP : Digest Access Authentication", RFC 2617, January 1997.

[7] International Organization for Standards, "Data Elements and Interchange Formats - Information Interchange - Representation of Dates and Times", ISO 8601, June 1988.

[8] Borenstein, N., Freed, F., "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, November 1996.

[9] American National Standard for Data Encryption Algorithm (DEA). Standard ANSI X3.92, ANSI, 1981.

[10] Data Encryption Standard, FIPS46-2, December 30, 1993.

[11] DES Modes of Operation, FIPS81, December 2, 1980

[12] IEEE/ANSI Std. 1003.2-1992, Information Technology – Portable Operating System Interface (POSIX®) Part 2

[13] Berners-Lee et al., "Uniform Resource Identifiers (URI): Generic Syntax", RFC 2396, IETF 1998

Index of Compliance Items

V

Index